



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1998-03

Theater Ballistic Missile defense-multisensor fusion, targeting and tracking techniques

San Jose, Antonio P.

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/32730>

Downloaded from NPS Archive: Calhoun



<http://www.nps.edu/library>

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



19980611 064

THEATRE BALLISTIC MISSILE DEFENSE- MULTISENSOR FUSION, TARGETING AND TRACKING TECHNIQUES

by

Antonio P. San Jose

March 1998

Thesis Advisor:
Second Reader:

Robert G. Hutchins
Harold A. Titus

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 1998		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE THEATER BALLISTIC MISSILE DEFENSE - MULTISENSOR FUSION, TARGETING AND TRACKING TECHNIQUES			5. FUNDING NUMBERS	
6. AUTHOR(S) Antonio P. San Jose				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The Gulf War illustrated how important ballistic missile defenses have become to the United States. The study of intercepting Theatre Ballistic Missiles (TBMs) in their boost phase was prompted by concerns about the widespread dissemination of submunitions and the differentiation of decoys from actual warheads released early in the missile's midcourse flight. Boost Phase Intercept (BPI) would alleviate this problem by destroying the enemy's ballistic missile in the missile's launch phase, thereby causing the lethal payload and debris from the engagement to fall back on the aggressor. This thesis focuses on the development of missile tracking algorithms to be used in the boost phase of TBMs. A missile encounters significant changes in velocity, acceleration, and direction during the boost phase, making it difficult to track. Extended Kalman filter (EKF), Alpha-Beta-Gamma filter, and Interacting Multiple Model (IMM) filtering techniques are developed to determine the missile tracking accuracy of TBMs during boost phase. Simulation results and actual TBM profiles from test data are presented to verify the tracking accuracy utilizing different filtering techniques.				
14. SUBJECT TERMS Kalman Filter, Alpha-Beta-Gamma Filter, Interacting Multiple Models, TBMD			15. NUMBER OF PAGES 248	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18 298-102

Approved for public release; distribution is unlimited

**THEATER BALLISTIC MISSILE DEFENSE -MULTISENSOR FUSION, TARGETING
AND TRACKING TECHNIQUES**

Antonio P. San Jose
Lieutenant, United States Navy
B.S., United States Naval Academy, 1990


Submitted in partial fulfillment
of the requirements for the degree of

**MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING**


from the

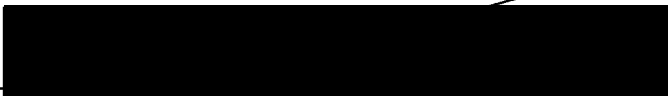
NAVAL POSTGRADUATE SCHOOL
March 1998

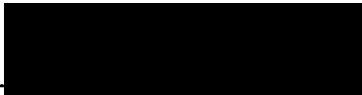
Author:


Antonio P. San Jose

Approved by:


Robert G. Hutchins, Thesis Advisor


Harold A. Titus, Second Reader


for Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The Gulf War illustrated how important ballistic missile defenses have become to the United States. The study of intercepting Theatre Ballistic Missiles (TBMs) in their boost phase was prompted by concerns about the widespread dissemination of submunitions and the differentiation of decoys from actual warheads released early in the missile's midcourse flight. Boost Phase Intercept (BPI) would alleviate this problem by destroying the enemy's ballistic missile in the missile's launch phase, thereby causing the lethal payload and debris from the engagement to fall back on the aggressor. This thesis focuses on the development of missile tracking algorithms to be used in the boost phase of TBMs. A missile encounters significant changes in velocity, acceleration, and direction during the boost phase, making it difficult to track. Extended Kalman filter (EKF), Alpha-Beta-Gamma filter, and Interacting Multiple Model (IMM) filtering techniques are developed to determine the missile tracking accuracy of TBMs during boost phase. Simulation results and actual TBM profiles from test data are presented to verify the tracking accuracy utilizing different filtering techniques.

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BALLISTIC MISSILE DEFENSE.....	1
B.	BOOST PHASE INTERCEPT	1
C.	THESIS ORGANIZATION	2
II.	BALLISTIC MISSILE TRAJECTORY.....	5
A.	GENERATING THE BALLISTIC MISSILE BASE TRAJECTORY.....	5
B.	RUNNING THE SIMULATION	9
C.	ADDING MEASUREMENT NOISE	15
III.	EXTENDED KALMAN FILTER.....	19
A.	DISCRETE TIME KALMAN FILTER	19
B.	EXTENDED KALMAN FILTER	21
C.	EKF IN TARGET TRACKING	26
D.	SIMULATION RESULTS	30
IV.	FIXED-COEFFICIENT FILTERING.....	39
A.	ALPHA-BETA-GAMMA TRACKER	39
B.	SIMULATION RESULTS	43
V.	INTERACTING MULTIPLE MODEL ALGORITHM	55
A.	IMM ALOGRITHM.....	55
B.	SIMULATION RESULTS	70

VI. ACTUAL TBM PROFILES	87
A. TBM PROFILES	87
B. TBM PROFILE 1	89
C. TBM PROFILE 4	102
D. TBM PROFILE 5	116
E. COMPARISON OF TBM PROFILES	129
VII. CONCLUSION	133
APPENDIX A. SOURCE CODE FOR BALLISTIC MISSILE SIMULATION.....	135
APPENDIX B. SOURCE CODE FOR EKF TRACKING ALGORITHM	141
APPENDIX C. SOURCE CODE FOR ALPHA-BETA-GAMMA TRACKING ALGORITHM.....	149
APPENDIX D. SOURCE CODE FOR IMM TRACKING ALGORITHM	157
APPENDIX E. TBM PROFILES	169
APPENDIX F. MATLAB® INFORMATION.....	235
LIST OF REFERENCES.....	237
INITIAL DISTRIBUTION LIST	239

I. INTRODUCTION

A. BALLISTIC MISSILE DEFENSE

The Gulf War illustrated how important ballistic missile defenses have become to the United States. The Iraqi use of theater ballistic missiles (TBMs) focused the United States defense on the danger posed by the widespread proliferation of TBMs. Today, over thirty countries possess ballistic missiles and more than twenty-five are believed to be developing nuclear, chemical, or biological weapons [Ref. 1]. Many of those same countries may be converting these weapons of mass destruction into warheads that can be delivered by ballistic missiles. Because of worldwide development efforts to increase the exportable supply of TBMs, missiles of increased range and payload will find their way into the weapons inventories of many nations during the next decade. Potential aggressors will have a potent capability to deliver short notice or surprise attacks that might threaten regional balances, U.S. allies, U.S. forces deployed overseas, and potentially U.S. territory. The ability to put a nuclear, chemical or biological warhead on a ballistic missile, along with the increasing ability to export such missiles, highlights the necessity for the United States to develop effective theater missile defense (TMD) systems. [Ref. 2, 3, 4]

B. BOOST PHASE INTERCEPT

The study of intercepting TBMs in the boost phase was prompted by concerns about the widespread dissemination of submunitions and the differentiation of decoys

from actual warheads released early in the midcourse phase. Boost Phase Intercept (BPI) would alleviate this problem by destroying the enemy's ballistic missile in the missile's initial launch phase, causing the lethal payload and the debris from the engagement to fall back on the aggressor. Because boost phase defenses intercept a missile prior to the release of its payload, BPI appears to be the only way to defend against submunitions. An advantage of the boost-phase defense is that during a launch, the missile's rocket motors spew out hot gases that are easy to locate; unfortunately, the motors burn for only a few minutes. The challenge of BPI lies in the ability to detect launch of the missile, to track it long enough to get a fix on its trajectory, and then to intercept it. All of this must be done in only a few minutes. The creation of a successful BPI would considerably ease the burden of relying solely on existing terminal defenses to combat TBMs. [Ref. 2]

C. THESIS ORGANIZATION

This thesis focuses on the development of missile tracking algorithms to be used in the boost phase of TBMs. Chapter II furnishes the reader with a basic understanding of generating a ballistic missile simulation. Chapter III provides background information on the Extended Kalman Filter (EKF) and discusses its use in missile tracking. Chapter IV provides background information on fixed-coefficient filtering, and discusses the development of the Alpha-Beta-Gamma filter used in missile tracking. Chapter V discusses the Interacting Multiple Model (IMM) algorithm in which multiple filter models are used to produce a combined position estimate. Chapter VI studies the implementation of the EKF, the Alpha-Beta-Gamma tracker, and the IMM algorithm on

actual TBM profiles. Chapter VII presents conclusions and recommendations for follow-on studies.

II. BALLISTIC MISSILE TRAJECTORY

This chapter provides background information so the reader has an understanding of the ballistic missile base trajectory used in the missile tracking algorithms presented in Chapters III, IV and V. A base trajectory is developed using flat earth equations of motion, which are modeled in SIMULINKTM. To simulate a sensor platform observing the missile, measurement noise with uncertainties in range, bearing and elevation is added to this base trajectory. The tracking algorithms are then implemented on these position measurements and the resulting filtered trajectory is compared to the base trajectory (used as true missile position) to determine the accuracy of our tracking algorithms.

A. GENERATING THE BALLISTIC MISSILE BASE TRAJECTORY

The ballistic missile base trajectory is generated using SIMULINKTM. The initialization file, *PtMissileInit.m*, initializes the following variables in order to generate a simulated ballistic missile trajectory:

- The missile is launched from the surface of the earth (0 km along the z axis), 30 km along the x axis, and 40 km along the y axis.
- The missile thrust (T) is approximately 6 gs.
- The missile's booster cut-off (tT_{off}) occurs 60 seconds after launch.

- The missile rolls approximately 40 degrees in elevation (*wel*) and 15 degrees in azimuth (*waz*), 20 seconds after launch.
- The coefficient of friction (*cfric*) is 0.5.
- The simulation sampling interval (*sinterval*) is 0.1 seconds.
- The missile is assumed to have a constant mass.
- The force of gravity (*g*) is assumed to be constant throughout the simulation.

After initialization, the SIMULINK™ model, *FlatEPtMissileSim.m*, is used to generate the ballistic missile simulation. *FlatEPtMissileSim.m* is shown in Figure 2.1. The SIMULINK™ model uses the following simulation parameters:

- Runge-Kutta 5 integration algorithm
- Minimum step size = 10^{-5}
- Maximum step size = 10^{-1}
- Relative error = 10^{-3}

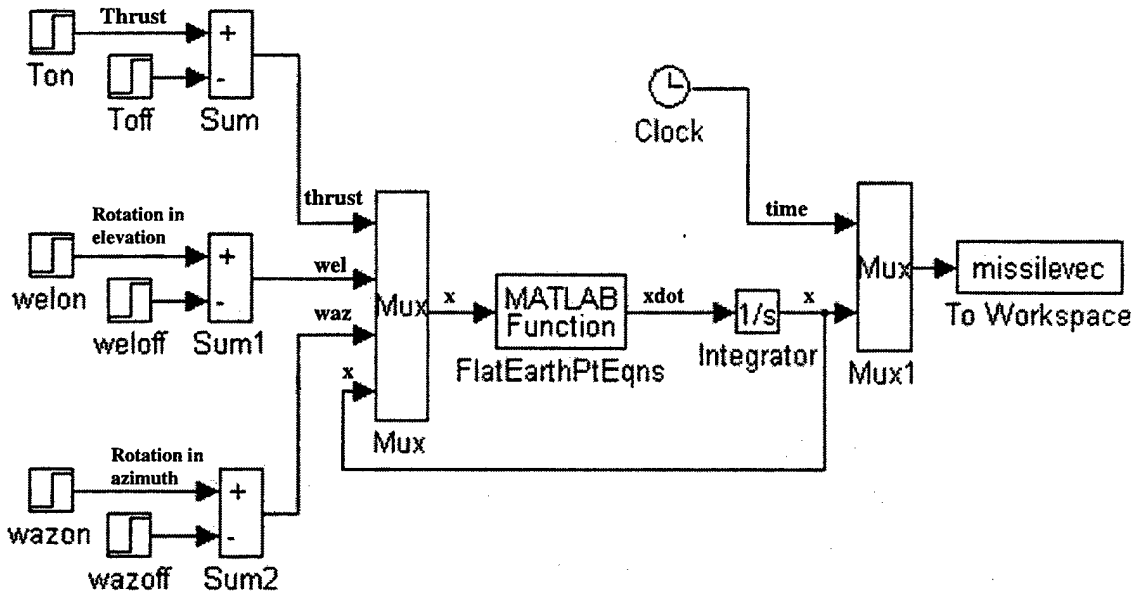


Figure 2.1 SIMULINK™ Model, FlatEPtMissileSim.m.

Within the SIMULINK™ model, the MATLAB® function, *FlatEarthPtEqns.m*, generates the missile dynamics using flat earth equations of motion, as outlined in *Aircraft Control and Simulation* [Ref. 5]. In addition, the atmospheric density is modeled in accordance with *Tactical and Strategic Missile Guidance* [Ref. 6], and is described as follows,

- Altitudes above 9144 meters: $\rho = 1.75228763 \times e^{-\frac{h}{6705.6}} \frac{kg}{m^3}$
- Altitudes below 9144 meters: $\rho = 1.22557 \times e^{-\frac{h}{9144}} \frac{kg}{m^3}$
- Altitudes below 0 meters (travel inside the earth's surface): $\rho = 100 \frac{kg}{m^3}$

In the SIMULINK™ model, the inputs to the missile dynamics function are thrust, rotation in elevation, rotation in azimuth, and the state vector, x . The missile state vector gives the missile's position, velocity, and acceleration data at each sampling interval of time. The missile state vector x , at time t_k , is defined as,

$$x_k = \begin{bmatrix} x - \text{position} \\ x - \text{velocity} \\ x - \text{acceleration} \\ y - \text{position} \\ y - \text{velocity} \\ y - \text{acceleration} \\ z - \text{position} \\ z - \text{velocity} \\ z - \text{acceleration} \end{bmatrix} = \begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \\ z \\ v_z \\ a_z \end{bmatrix} \quad (2.1)$$

with

$$\dot{x} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \\ \ddot{y} \\ \dot{z} \\ \ddot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ a_x \\ 0 \\ v_y \\ a_y \\ 0 \\ v_z \\ a_z - g \\ 0 \end{bmatrix} \quad (2.2)$$

The missile state vector is generated every 0.1 seconds, and the resulting data is stored in the MATLAB® workspace under the variable *missilevec*.

B. RUNNING THE SIMULATION

The following steps are used to run the ballistic missile simulation:

- **STEP 1.** In the MATLAB[®] workspace, run the initialization file, *PtMissileInit.m*.
- **STEP 2.** In the SIMULINK[™] workspace, open the SIMULINK[™] model, *FlatEPtMissileSim.m*, and configure the simulation parameters as described above.
- **STEP 3.** Start the simulation in SIMULINK[™].
- **STEP 4.** Graph the output by running the plotting program, *FlatEPTPlots.m*, in the MATLAB[®] workspace.

The resulting plots of the simulation are shown in Figures 2.2(a) through (i). Figures 2.2(a) through (g) give the reader a visual representation of the ballistic missile base trajectory. Figures 2.2(h) and (i) emphasize the missile in its boost phase. The MATLAB[®] source codes for initialization, missile dynamics and plotting are provided in Appendix A.

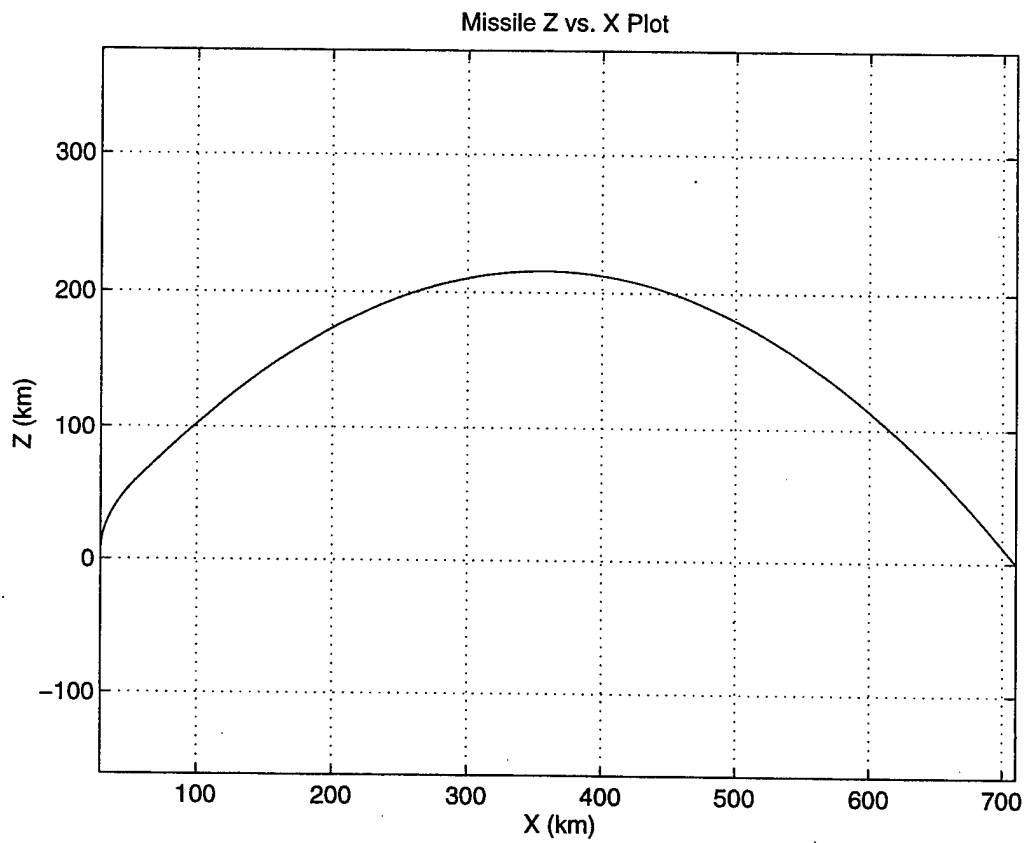


Figure 2.2(a) Missile Z vs. X Plot.

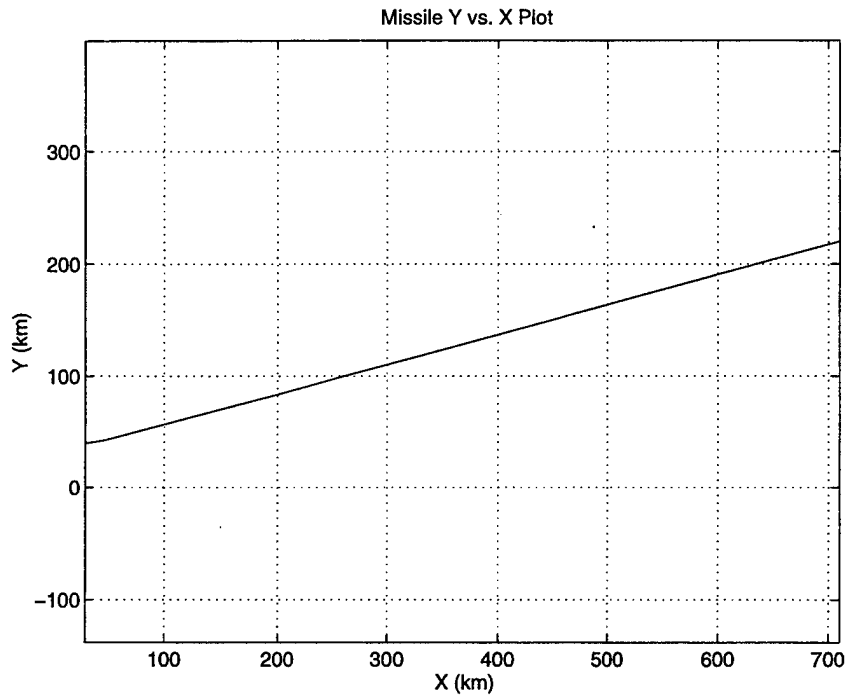


Figure 2.2(b) Missile Y vs. X Plot.

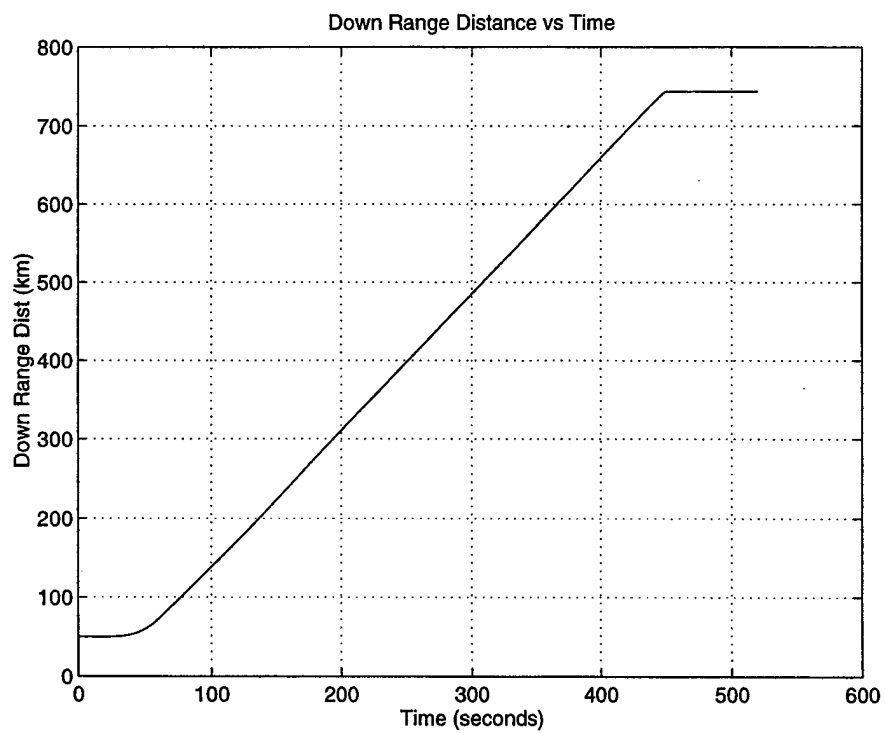


Figure2.2(c) Missile Downrange Distance vs. Time.

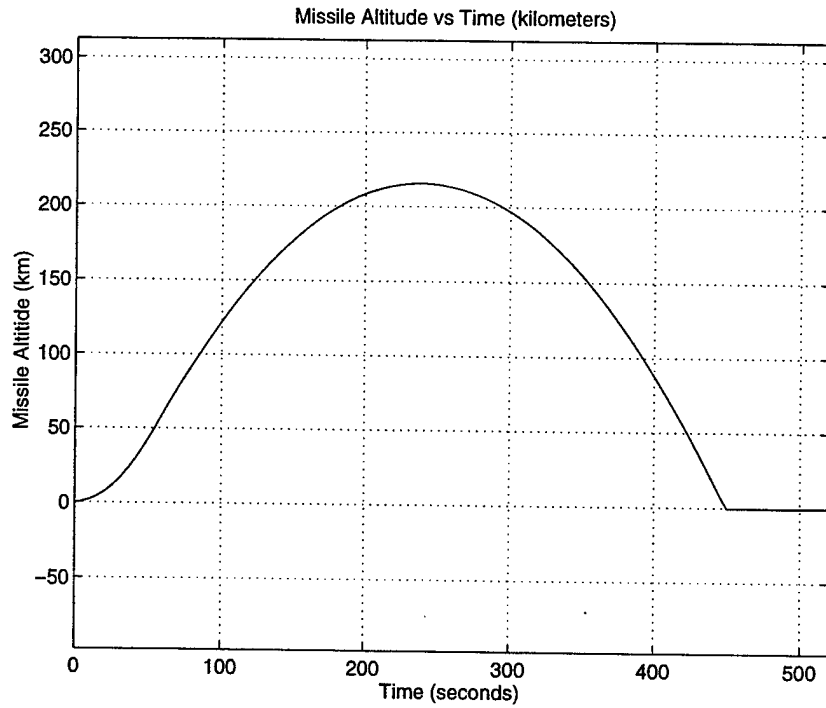


Figure 2.2(d) Missile Altitude vs. Time.

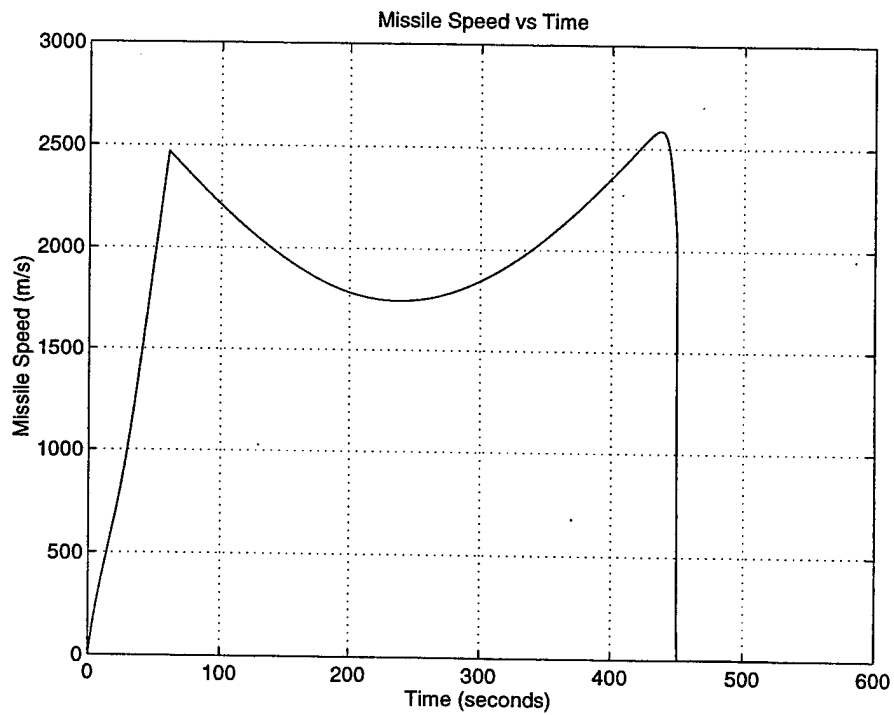


Figure 2.2(e) Missile Speed vs. Time.

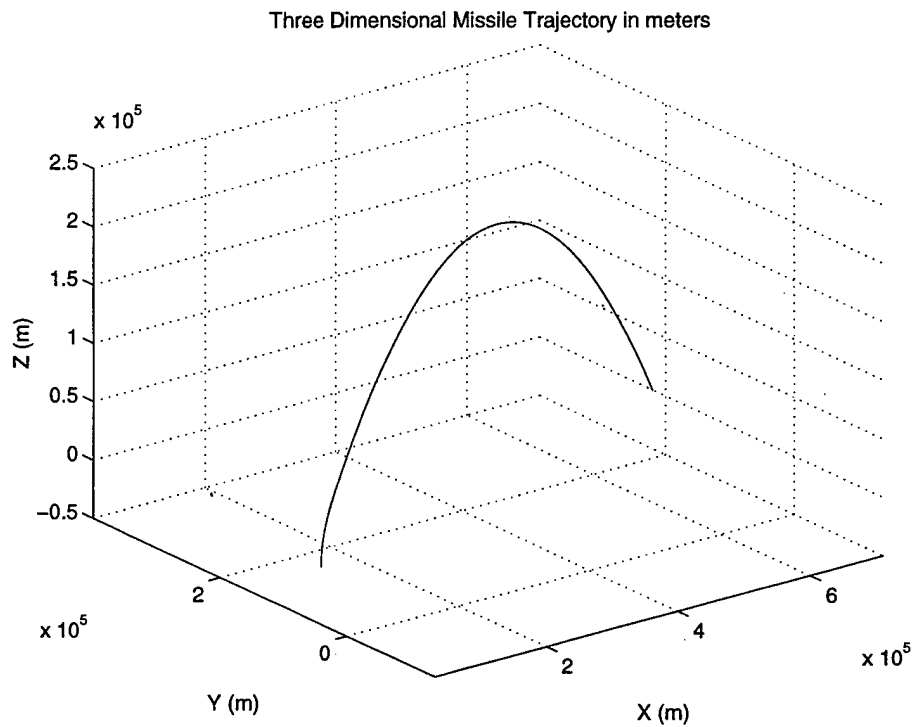


Figure 2.2(f) Three Dimensional Missile Trajectory in meters.

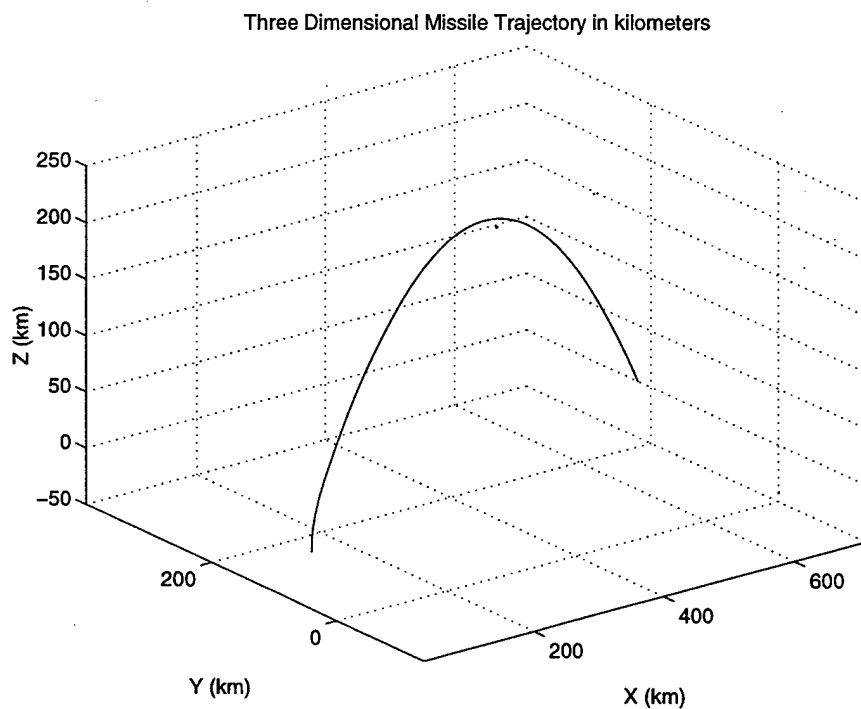


Figure 2.2(g) Three Dimensional Missile Trajectory in kilometers.

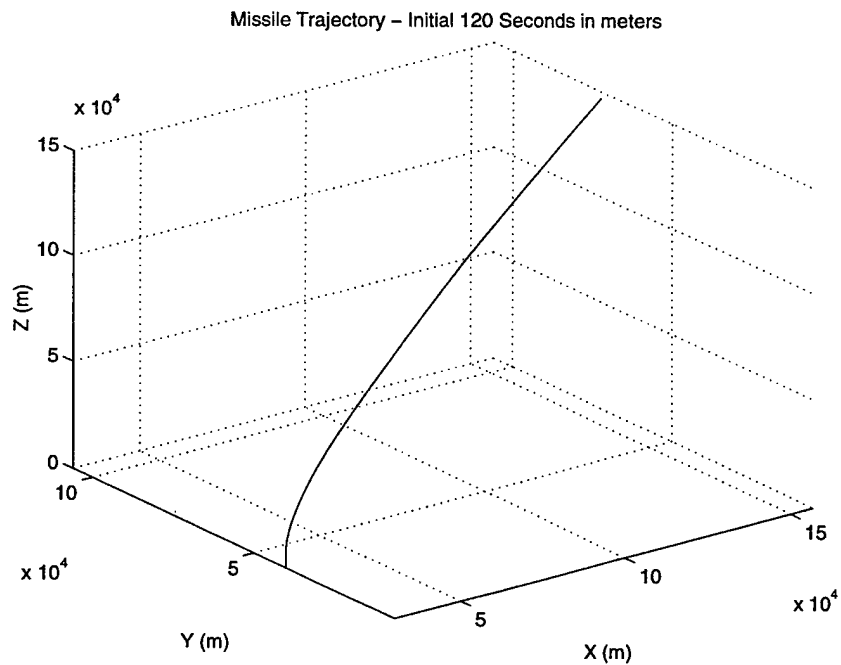


Figure 2.2(h) Missile Launch (close-up), Initial 120 Seconds (in meters).

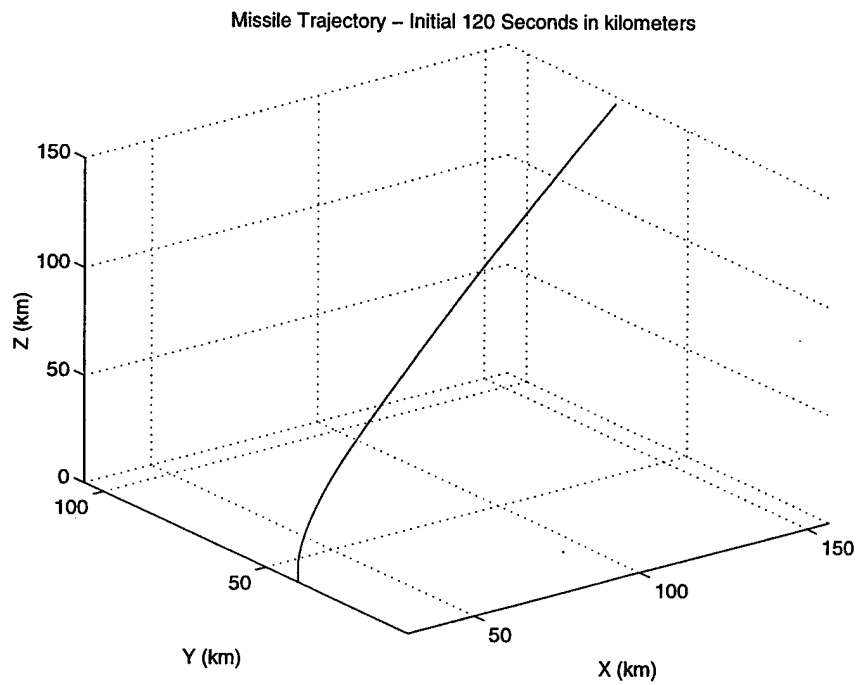


Figure 2.2(i) Missile Launch (close-up), Initial 120 Seconds (in kilometers).

C. ADDING MEASUREMENT NOISE

A surface ship is selected as the sensor platform to observe the missile. The location is chosen to be 100 km in the x direction, 100 km in the y direction, and 0 km in the z direction. The sensor position is marked by an 'x', and its position relative to the missile trajectory is shown in Figures 2.3(a) and (b). The surface platform observes the missile's position through measurements in range, bearing and elevation (i.e. radar measurements). To account for the inaccuracies of the sensor's measurements, measurement noise with uncertainties in range, bearing, and elevation is added to the base trajectory. During this study, the measurement noise in the tracking algorithms is chosen to have the following standard deviations:

- $\sigma_{\text{range}} = 10 \text{ meters}$
- $\sigma_{\text{bearing}} = 1^\circ$
- $\sigma_{\text{elevation}} = 1^\circ$

Figure 2.3(a) shows the boost phase of the ballistic missile base trajectory. Figure 2.3(b) shows the same trajectory with the addition of measurement noise.

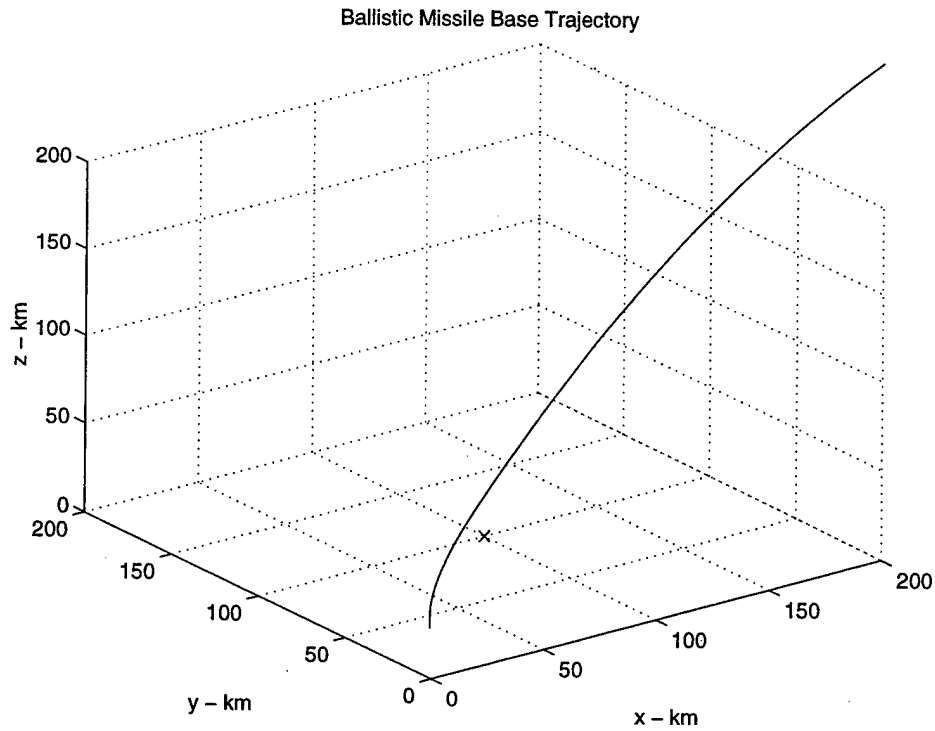


Figure 2.3(a) Ballistic Missile Base Trajectory.

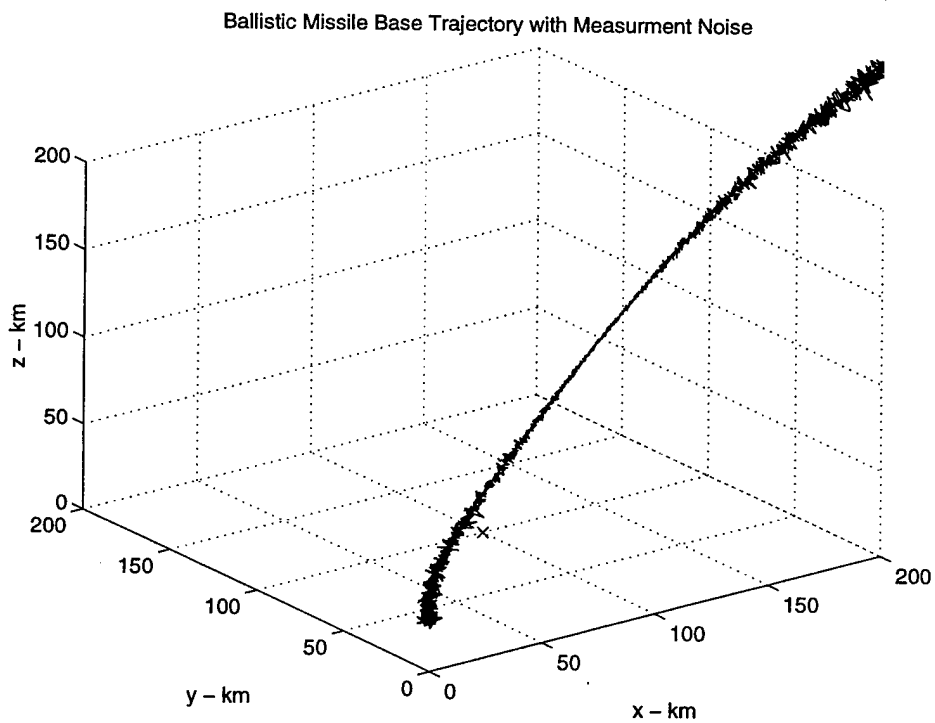


Figure 2.3(b) Ballistic Missile Base Trajectory with Measurement Noise.

Chapter III begins the investigation on ballistic missile tracking during the boost phase. The missile tracking algorithms focus on the boost phase, therefore only the initial 120 seconds of the simulated missile data are examined. Chapter III also provides background information on the Extended Kalman Filter, and describes the tracking algorithm in detail. Simulation results are presented and compared to the base trajectory developed in this chapter to determine the accuracy of the tracking algorithms.

III. EXTENDED KALMAN FILTER

This chapter provides background information on the development of a tracking algorithm utilizing the Extended Kalman Filter (EKF) equations. The discrete time Kalman filter equations are briefly discussed to familiarize the reader with the Kalman filter before presenting the more advanced EKF equations in the following sections, and before presenting the Interacting Multiple Model equations in Chapter V. In this chapter, an EKF tracking algorithm is developed and implemented on the position measurements of the ballistic missile base trajectory developed in Chapter II. Simulation results are presented and the EKF tracking accuracy is analyzed.

A. DISCRETE TIME KALMAN FILTER

The purpose of the Kalman filter is to estimate a state vector at the time of the last measurement based on the knowledge of all past measurements. When used in missile tracking, the Kalman filter equations are used to estimate present and future target kinematic quantities such as: positions, velocities, and accelerations. First assume that the missile dynamic process is modeled in discrete form as follows,

$$x_{k+1} = F_k x_k + \omega_k \quad (3.1)$$

where x_k is the n dimensional missile state vector that includes quantities to be estimated, F_k is the known state transition matrix, and ω_k is the plant noise associated with the target. The plant noise, ω_k , is assumed to be zero mean (implies an unbiased sensor), white and Gaussian with known covariance Q_k . The measurement process is as follows:

$$z_k = H_k x_k + v_k \quad (3.2)$$

where the measurements are linear combinations of the state variables, which are corrupted by the addition of uncorrelated measurement noise, v . The variable z_k designates the sensor measurement at time, t_k . The matrix H_k is a constant matrix related to the number of dimensions being observed. As in the plant noise above, the measurement noise, v_k , is assumed to be zero mean, white and Gaussian with known covariance R_k . [Ref. 7]

To start the Kalman algorithm, the initial state estimate, \hat{x}_0 , and its associated covariance, P_0 , are assumed to be known a priori. The algorithm starts a recursive process, in which it loops sequentially over the measurement, and then updates the measurement at each measurement time. The process of *updating* the state estimate when a new measurement is obtained can be broken down into two steps: *prediction* and *correction*. *Prediction* refers to the estimation of the state vector to the next measurement time. In this process, the state estimate and associated covariance are predicted to the next measurement time using the following prediction equations,

$$\hat{x}_{k+1|k} = F_k x_{k|k} + \omega_k \quad (3.3)$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k \quad (3.4)$$

where T denotes transpose. *Correction* refers to updating (or correcting) the state estimate and associated covariance based on the new measurement, using the following correction equations,

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1} [\tilde{z}_{k+1}] \quad (3.5)$$

where K_{k+1} (Kalman Gain) and \tilde{z}_{k+1} (residual vector) are defined as

$$K_{k+1} = P_{k+1|k} H_{k+1}^T [H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1}]^{-1} \quad (3.6)$$

$$\tilde{z}_{k+1} = z_{k+1} - H_{k+1} \hat{x}_{k+1|k} \quad (3.7)$$

The covariance update equation is

$$P_{k+1|k+1} = (I - K_{k+1} H_{k+1}) P_{k+1|k} \quad (3.8)$$

where I is the identity matrix. An equivalent covariance update equation is

$$P_{k+1|k+1} = (I - K_{k+1} H_{k+1}) P_{k+1|k} (I - K_{k+1} H_{k+1})^T + K_{k+1} R_{k+1} K_{k+1}^T \quad (3.9)$$

It is referred to as the Joseph Form, and is used in throughout this study because it behaves better numerically in computer calculations [Ref. 8]. The combined set of *prediction* and *correction* equations constitutes the discrete time Kalman filter. The preceding information is provided as a link to understand the development of the EKF tracking algorithm. [Ref. 9, 10, 11]

B. EXTENDED KALMAN FILTER

In applications involving nonlinear dynamics or nonlinear measurement relationships, the EKF, vice the traditional Kalman filter (as described in the previous section), is generally used. In this study, the measurement relationships from the sensor (radar measurements in range, bearing and elevation) are nonlinear; therefore, the EKF is used in our ballistic missile tracking algorithm. Because the basic equations in the EKF are similar to that of the traditional Kalman filter, an understanding of the traditional Kalman filter is essential. The main difference between the EKF and the Kalman filter is

the evaluation of the Jacobians of the state transition and the measurement equations (the partial derivatives of the F and H matrices) [Ref. 9]. This difference will be highlighted again in the following derivation of the EKF equations.

In a system with nonlinearities in the dynamics or the measurement process, it is desirable to have the same framework as in a linear system. Assume the following nonlinear system equations,

$$x_{k+1} = f_k(x_k) + \omega_k \quad (3.10)$$

$$z_k = h_k(x_k) + v_k \quad (3.11)$$

where $f_k(x_k)$ is the nonlinear dynamics equation, and $h_k(x_k)$ is the nonlinear measurement equation. The noise processes v_k and ω_k , are assumed to be white (uncorrelated)

Gaussian processes and mutually independent. Hence,

$$E[v_k] = 0 \quad (3.12)$$

$$E[v_k v_l'] = Q_k \cdot \delta_{kl} \quad (3.13)$$

where δ_{kl} is the Kronecker delta function,

$$E[\omega_k] = 0 \quad (3.14)$$

$$E[\omega_k \omega_l'] = R_k \cdot \delta_{kl} \quad (3.15)$$

with no cross correlation such that

$$0 = E[v_k \omega_l'] = E[v_k x_o'] = E[\omega_k x_o'] \quad \forall k, l \quad (3.16)$$

In order to determine the EKF prediction and correction equations, the nonlinear system of equations ($f_k(x_k)$ and $h_k(x_k)$) must first be linearized. The linearization is obtained by a series expansion of the nonlinear dynamics and of the nonlinear measurement equations.

To obtain the predicted state $\hat{x}_{k+1|k}$, the nonlinear function is expanded in a Taylor series around the latest estimate, $\hat{x}_{k|k}$, with terms up to the first order to obtain a first order EKF. The first order Taylor series expansions are required for the dynamic process and for the measurement process, and thus the matrices F_k and H_k must be determined. We define F_k as the gradient of f_k evaluated at the most recent estimate, $\hat{x}_{k|k}$,

$$F_k \equiv \left. \frac{\partial f_k(x)}{\partial x} \right|_{x=\hat{x}(k|k)} \quad (3.17)$$

and H_k as the gradient of h_k evaluated at the most recent estimate, $\hat{x}_{k|k}$,

$$H_k \equiv \left. \frac{\partial h_k(x)}{\partial x} \right|_{x=\hat{x}(k|k-1)} \quad (3.18)$$

The Taylor series expansions about the estimates are as follows,

$$f_k(x_k) = f_k(\hat{x}_{k|k}) + F_k(x_k - \hat{x}_{k|k}) + \dots \quad (3.19)$$

$$h_k(x_k) = h_k(\hat{x}_{k|k-1}) + H_k(x_k - \hat{x}_{k|k-1}) + \dots \quad (3.20)$$

Then, the approximate system equations, neglecting the higher order terms are,

$$\begin{aligned} x_{k+1} &= f_k(x_k) + \omega_k \\ &= \left(f_k(\hat{x}_{k|k}) + F_k(x_k - \hat{x}_{k|k}) \right) + \omega_k \\ &= F_k x_k + f_k(\hat{x}_{k|k}) - F_k \hat{x}_{k|k} + \omega_k \end{aligned} \quad (3.21)$$

$$\begin{aligned}
z_k &= h_k(x_k) + v_k \\
&= \left(h_k(\hat{x}_{k|k-1}) + H_k(x_k - \hat{x}_{k|k-1}) \right) + v_k \\
&= H_k x_k + h_k(\hat{x}_{k|k}) - H_k \hat{x}_{k|k} + v_k
\end{aligned} \tag{3.22}$$

Hence, the approximate (linearized) system of equations are,

$$x_{k+1} = F_k x_k + \omega_k + u_k \tag{3.23}$$

$$z_k = H_k x_k + v_k + y_k \tag{3.24}$$

with the deterministic terms

$$u_k \equiv f_k(\hat{x}_{k|k}) - F_k \hat{x}_{k|k} \tag{3.25}$$

$$y_k \equiv h_k(\hat{x}_{k|k-1}) - H_k \hat{x}_{k|k-1} \tag{3.26}$$

The Kalman filter *prediction* and *correction* steps for these approximate equations are as follows:

Prediction: In the state estimate, substitute \hat{x} for x , include the deterministic terms and drop the zero mean noise.

$$\begin{aligned}
\hat{x}_{k+1|k} &= F_k \hat{x}_{k|k} + u_k \\
&= F_k \hat{x}_{k|k} + \left[f_k(\hat{x}_{k|k}) - F_k \hat{x}_{k|k} \right] \\
&= f_k(\hat{x}_{k|k})
\end{aligned} \tag{3.27}$$

The covariance prediction is a linear Gaussian update of the noise terms,

$$P_{k+1|k} = F_k P_{k|k} (F_k)^T + Q_k \tag{3.28}$$

Correction:

$$\begin{aligned}\hat{z}_{k|k-1} &= H_k \hat{x}_{k|k-1} + y_k \\ &= H_k \hat{x}_{k|k-1} + \left[h_k(\hat{x}_{k|k-1}) - H_k \hat{x}_{k|k-1} \right] \\ &= h_k(\hat{x}_{k|k-1})\end{aligned}\tag{3.29}$$

Hence, the state update equation is,

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k [\tilde{z}_k]\tag{3.30}$$

with

$$\tilde{z}_k \equiv z_k - \hat{z}_{k|k-1}\tag{3.31}$$

and

$$K_k = P_{k|k-1} H_k^T \left[H_k P_{k|k-1} H_k^T + R \right]^{-1}\tag{3.32}$$

The covariance update equation using the gradient matrices is,

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}\tag{3.33}$$

with the equivalent Joseph form [Ref. 8],

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^T + K_k R_k K_k^T\tag{3.34}$$

These Kalman filter prediction and correction equations are exact for the set of approximate system equations. [Ref. 11]

C. EKF IN TARGET TRACKING

In this section, a ballistic missile tracking algorithm is developed utilizing the Extended Kalman Filter equations. In this algorithm, the system equations are the standard tracking equations,

$$x_{k+1} = F_k x_k + G_k + \omega_k \quad (3.35)$$

$$z_k = h_k x_k + v_k \quad (3.36)$$

where x_k is the missile state vector,

$$x_k = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \\ z \\ v_z \\ a_z \end{bmatrix} \quad (3.37)$$

F_k is the linear state transition matrix,

$$F_k = \begin{bmatrix} 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.38)$$

G_k is the gravity matrix, which accounts for the force of gravity in the z direction with

$$g = 9.8 \frac{\text{m}}{\text{s}^2},$$

$$G^1 = -g \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{\Delta^2}{2} \\ \Delta \\ 0 \end{bmatrix} \quad (3.39)$$

ω_k is the plant noise with covariance Q_k ,

$$Q_k = q^2 \times \begin{bmatrix} \frac{\Delta^5}{20} & \frac{\Delta^4}{8} & \frac{\Delta^3}{6} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\Delta^4}{8} & \frac{\Delta^3}{3} & \frac{\Delta^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\Delta^3}{6} & \frac{\Delta^2}{2} & \Delta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta^5}{20} & \frac{\Delta^4}{8} & \frac{\Delta^3}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta^4}{8} & \frac{\Delta^3}{3} & \frac{\Delta^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta^3}{6} & \frac{\Delta^2}{2} & \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta^5}{20} & \frac{\Delta^4}{8} & \frac{\Delta^3}{6} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta^4}{8} & \frac{\Delta^3}{3} & \frac{\Delta^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\Delta^3}{6} & \frac{\Delta^2}{2} & \Delta \end{bmatrix} \quad (3.40)$$

where Δ is the sampling interval and q^2 is a scaling factor used to account for unmodeled target maneuver accelerations, and v_k is measurement noise with covariance R_k ,

$$R = \begin{bmatrix} \sigma_{\text{range}}^2 & 0 & 0 \\ 0 & \sigma_{\text{bearing}}^2 & 0 \\ 0 & 0 & \sigma_{\text{elevation}}^2 \end{bmatrix} \quad (3.41)$$

with standard deviations as defined in Chapter II.

Although the missile dynamics in this system are linear, the measurement process is nonlinear. As discussed in Chapter II, the sensor observing the missile is assumed to be a surface platform located 100 km in the x direction, 100 km in the y direction and 0 km in the z direction. The surface platform observes the missile positions through measurements in range, bearing and elevation (radar measurements) relative to the sensor as shown below,

$$h_k = \begin{bmatrix} \text{range} \\ \text{bearing} \\ \text{elevation} \end{bmatrix} \quad (3.42)$$

where

$$\text{range} = \sqrt{x^2 + y^2 + z^2} = \sqrt{x_1^2 + x_4^2 + x_7^2} \quad (3.43)$$

$$\text{bearing} = \tan^{-1} \left[\frac{y}{x} \right] = \tan^{-1} \left[\frac{x_4}{x_1} \right] \quad (3.44)$$

$$r' = \sqrt{x^2 + y^2} = \sqrt{x_1^2 + x_4^2} \quad (3.45)$$

$$\text{elevation} = \tan^{-1} \left[\frac{z}{r'} \right] \quad (3.46)$$

These measurement equations are clearly nonlinear, and therefore the nonlinear measurement equations must be linearized using a series expansion of the measurement equation h_k . Applying the definition of the H_k matrix, as stated in Equation 3.18, the gradient of h_k is determined to be,

$$H_k = \begin{bmatrix} \frac{\partial r(x)}{\partial x_1} & \frac{\partial r(x)}{\partial x_2} & \frac{\partial r(x)}{\partial x_3} & \frac{\partial r(x)}{\partial x_4} & \frac{\partial r(x)}{\partial x_5} & \frac{\partial r(x)}{\partial x_6} & \frac{\partial r(x)}{\partial x_7} & \frac{\partial r(x)}{\partial x_8} & \frac{\partial r(x)}{\partial x_9} \\ \frac{\partial b(x)}{\partial x_1} & \frac{\partial b(x)}{\partial x_2} & \frac{\partial b(x)}{\partial x_3} & \frac{\partial b(x)}{\partial x_4} & \frac{\partial b(x)}{\partial x_5} & \frac{\partial b(x)}{\partial x_6} & \frac{\partial b(x)}{\partial x_7} & \frac{\partial b(x)}{\partial x_8} & \frac{\partial b(x)}{\partial x_9} \\ \frac{\partial e(x)}{\partial x_1} & \frac{\partial e(x)}{\partial x_2} & \frac{\partial e(x)}{\partial x_3} & \frac{\partial e(x)}{\partial x_4} & \frac{\partial e(x)}{\partial x_5} & \frac{\partial e(x)}{\partial x_6} & \frac{\partial e(x)}{\partial x_7} & \frac{\partial e(x)}{\partial x_8} & \frac{\partial e(x)}{\partial x_9} \end{bmatrix} \quad (3.47)$$

which simplifies to

$$H_k = \begin{bmatrix} \frac{x_1}{\sqrt{x_1^2 + x_4^2 + x_7^2}} & 0 & 0 & \frac{x_4}{\sqrt{x_1^2 + x_4^2 + x_7^2}} & 0 & 0 & \frac{x_7}{\sqrt{x_1^2 + x_4^2 + x_7^2}} & 0 & 0 \\ \frac{-x_4}{x_1^2 + x_4^2} & 0 & 0 & \frac{x_1}{x_1^2 + x_4^2} & 0 & 0 & 0 & 0 & 0 \\ \frac{-x_1 \cdot x_7}{\sqrt{x_1^2 + x_4^2}(x_1^2 + x_4^2 + x_7^2)} & 0 & 0 & \frac{-x_4 \cdot x_7}{\sqrt{x_1^2 + x_4^2}(x_1^2 + x_4^2 + x_7^2)} & 0 & 0 & \frac{\sqrt{x_1^2 + x_4^2}}{x_1^2 + x_4^2 + x_7^2} & 0 & 0 \end{bmatrix} \quad (3.48)$$

Therefore the approximate (linearized) system of equations are,

$$x_{k+1} = F_k x_k + G_k + \omega_k \quad (3.49)$$

$$z_k = H_k x_k + v_k + y_k \quad (3.50)$$

with deterministic terms

$$y_k \equiv h_k(\hat{x}_{k|k-1}) - H_k \hat{x}_{k|k-1} \quad (3.51)$$

The EKF tracking algorithm is implemented in MATLAB[®] by applying the matrices developed in this section to the EKF prediction and correction equations as outlined in Equations 3.27 through 3.34. Simulation results of the EKF algorithm are presented in the following section. The source code for the EKF algorithm is presented in Appendix B.

D. SIMULATION RESULTS

The EKF tracking algorithm is implemented on the ballistic missile base trajectory with added measurement noise. The results of the EKF tracking algorithm are obtained by running the EKF algorithm in MATLAB[®] and by plotting the average trajectories over 10 simulation runs, with $q^2 = 10$ and with the sampling interval (Δ) equal to 0.1 seconds. In order to get an accurate representation of the mean distance error, a graph of the mean distance error is obtained by running the EKF algorithm over 100 simulation runs. Figure 3.1(a) shows the ballistic missile base trajectory during boost phase. As stated in Chapter II, standard deviations in range, bearing and elevation were chosen as 10 meters, 1 degree, and 1 degree respectively, with the resulting measurement noise shown in Figure 3.1(b). The results of the EKF tracking algorithm are shown in Figures 3.2(a) through (c), which show a close up of the EKF trajectory at 40 seconds, 60 seconds and 80 seconds respectively. Figure 3.3 shows the EKF mean distance error throughout the boost phase. The top graph indicates the average distance error created by the measurement noise that is added to the base trajectory. The bottom graph indicates the distance error of the EKF tracking algorithm. When viewing this graph, it is evident

that the overall mean distance error is significantly reduced by approximately 75 percent; however, the EKF algorithm has difficulty tracking the missile in two distinct areas. During the first few seconds while the missile is accelerating and rolling over, the mean distance error peaks to approximately 600 meters. Secondly, at time 60 seconds, after the booster cut off, the missile changes from an accelerating model to a ballistic model at which the mean distance error peaks to a value of approximately 800 meters. The MATLAB[®] source code for the EKF tracking algorithm is provided in Appendix B.

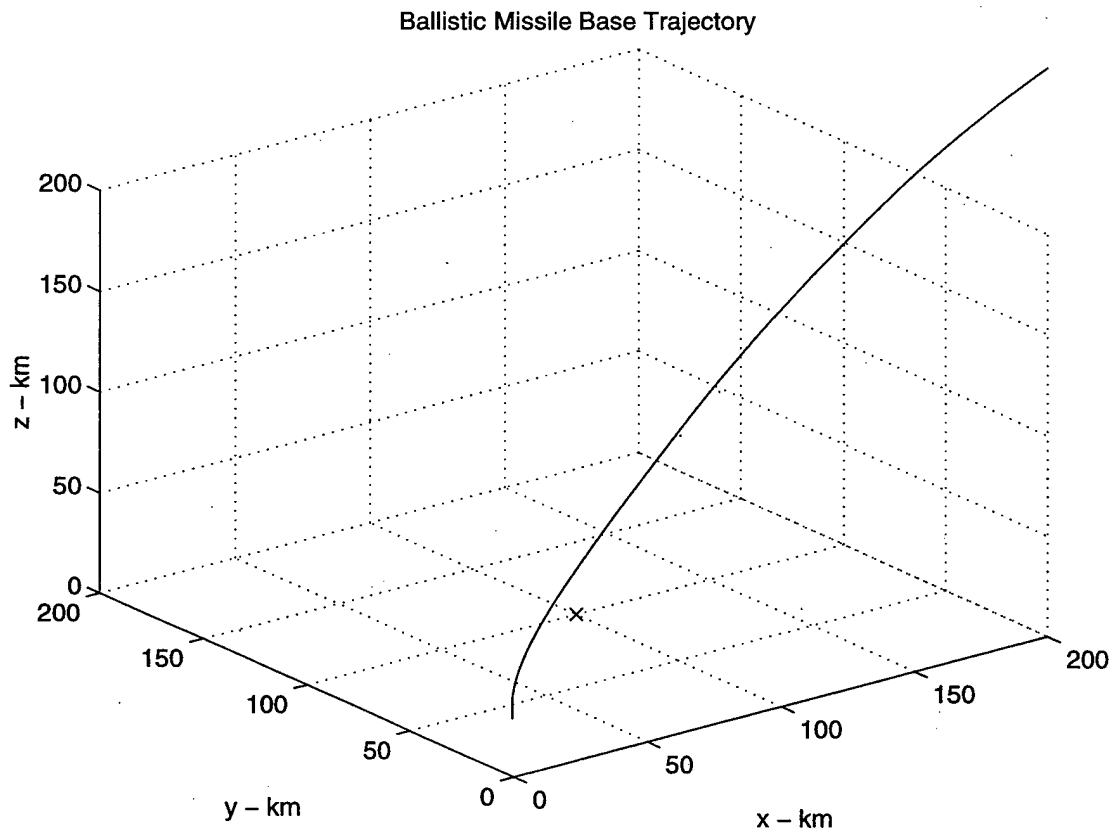


Figure 3.1(a) Ballistic Missile Base Trajectory.

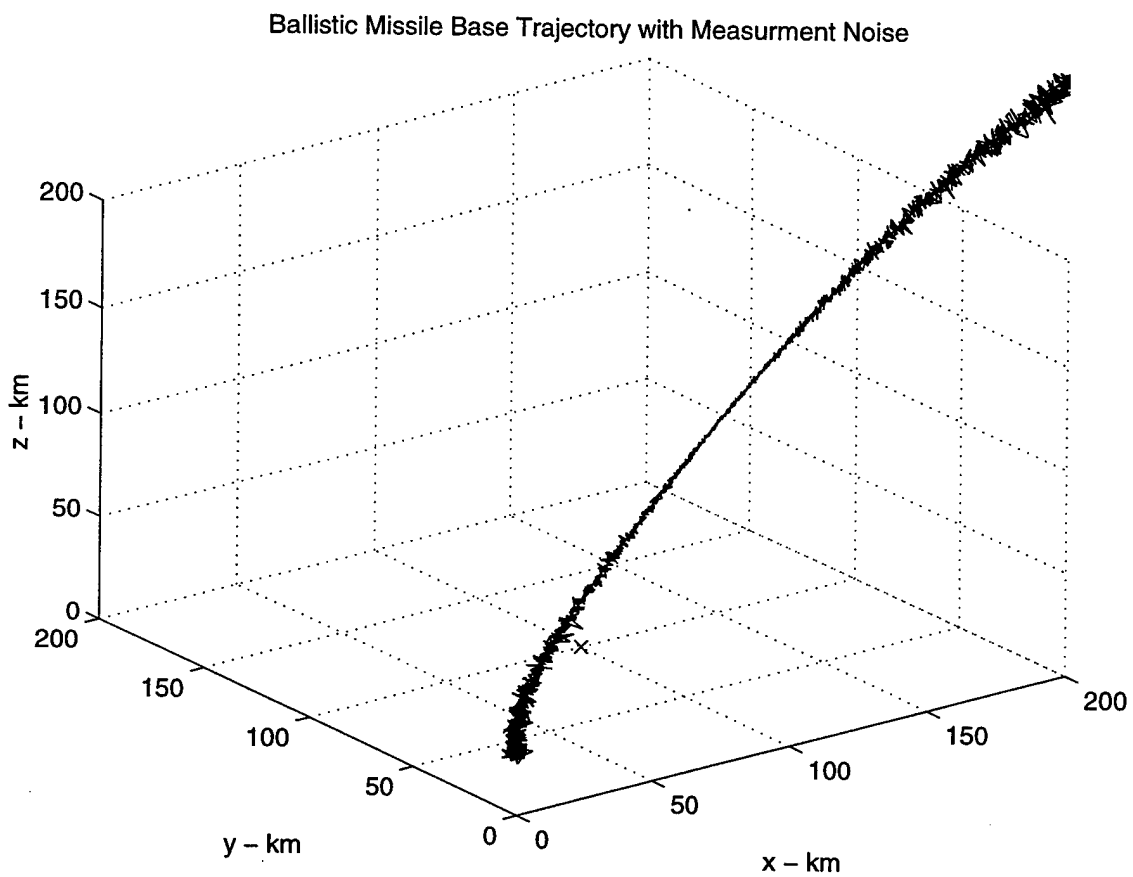


Figure 3.1(b) Ballistic Missile Base Trajectory with Measurement Noise.

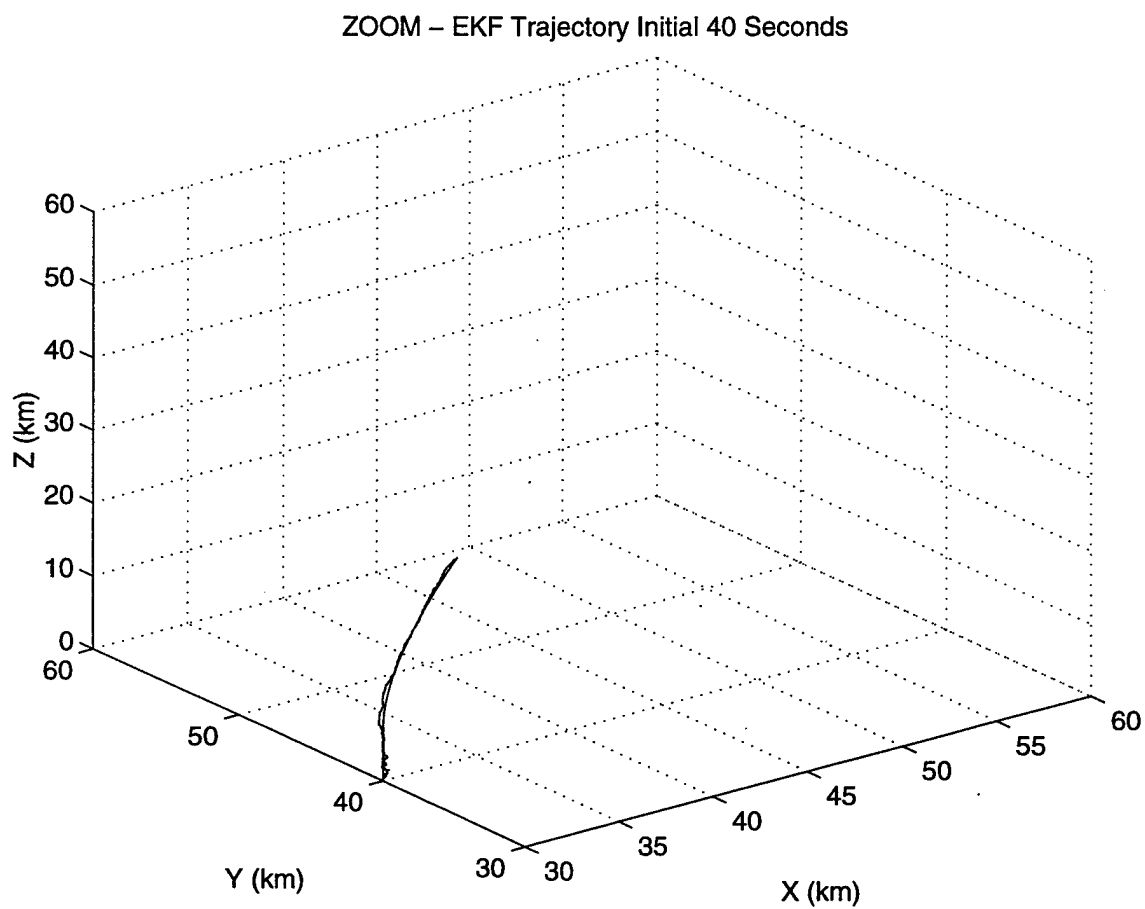


Figure 3.2(a) Close-up of the EKF Trajectory, Initial 40 seconds (10 Runs).

ZOOM – EKF Trajectory Initial 60 Seconds

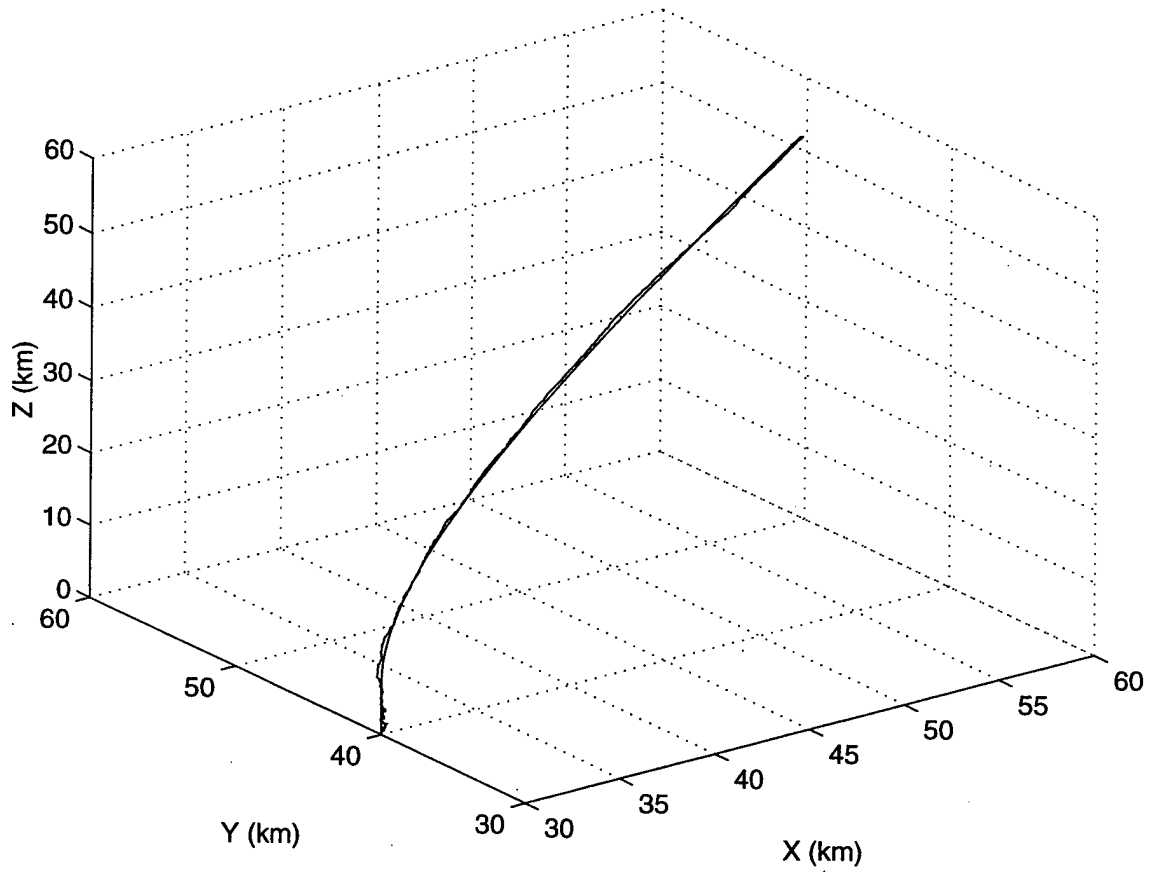


Figure 3.2(b) Close-up of the EKF Trajectory, Initial 60 seconds (10 Runs).

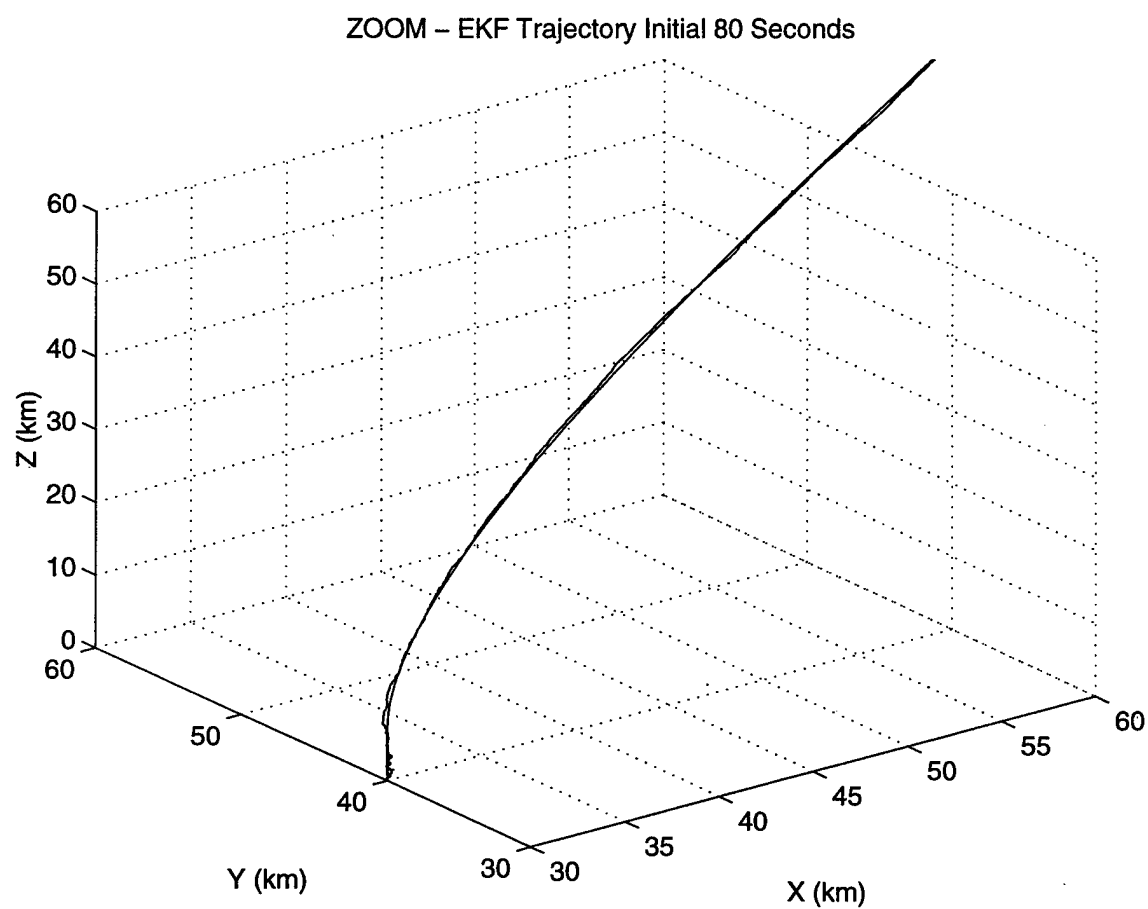


Figure 3.2(c) Close-up of the EKF Trajectory, Initial 80 seconds (10 Runs).

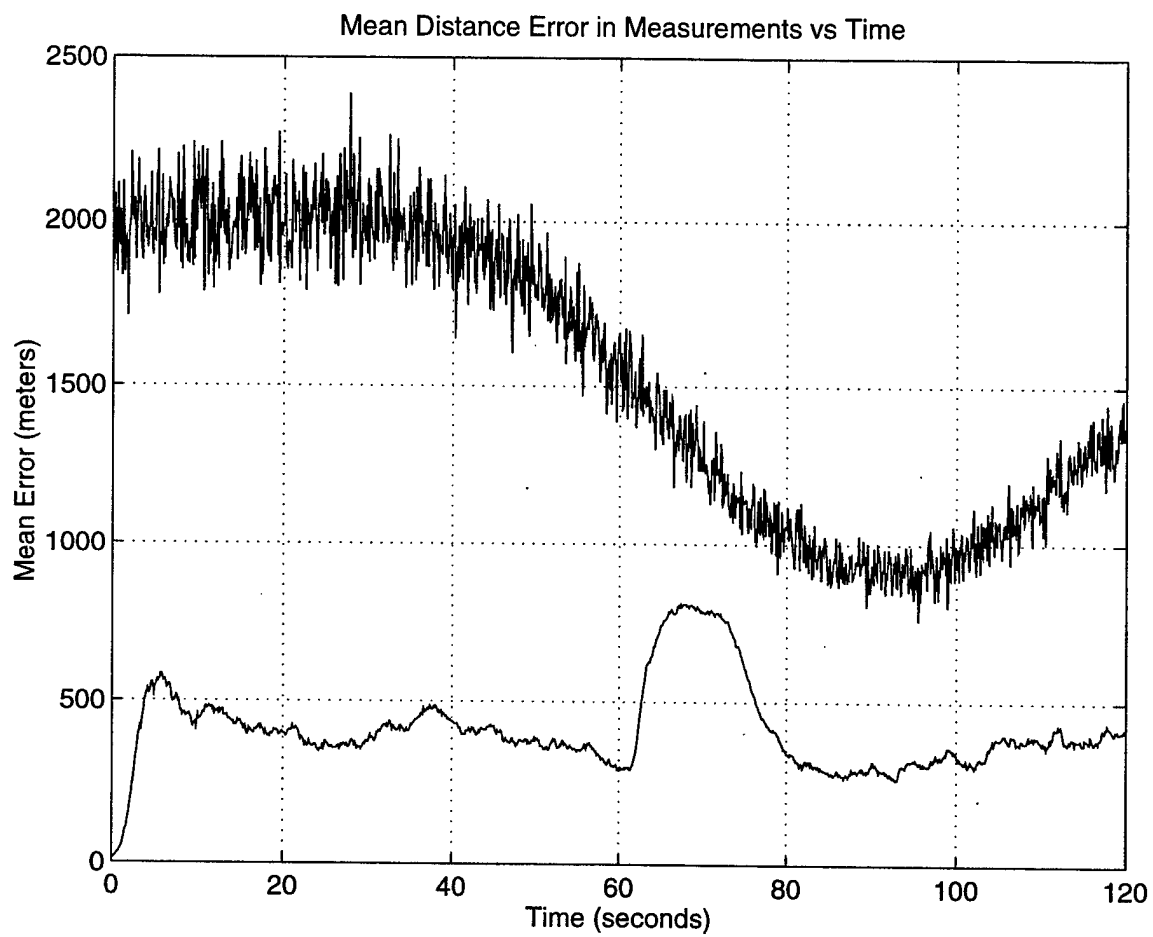


Figure 3.3 EKF Mean Distance Error (100 Runs).

In an attempt to reduce the tracking distance error, two other tracking algorithms are examined. Chapter IV investigates the constant gain, or fixed-coefficient, filter called the Alpha-Beta-Gamma tracker and determines its missile tracking capability. Simulation results are presented and compared to the EKF results in this section. Chapter IV investigates the missile tracking accuracy of a multiple model system using the Interacting Multiple Model (IMM) algorithm. Background information on the IMM algorithm is discussed and the simulation results are analyzed.

IV. FIXED-COEFFICIENT FILTERING

This chapter provides information on the development of a tracking algorithm that utilizes fixed-coefficient filtering. The advantage of this type of filter over the Kalman filter is its simple implementation where fixed parameters are used for filter gains. One of the most commonly used fixed-coefficient (constant gain) filters is the Alpha-Beta-Gamma (α - β - γ) tracker. The α - β - γ tracker is a constant gain filter used specifically in tracking systems when position measurements are available and when the state vector consists of positions, velocities, and accelerations. The actual nature of the noise processes, including the covariance matrices, Q and R , are not required, thus simplifying the filter design. The α - β - γ filter equations are presented and the developed tracking algorithm is implemented on the position measurements of the ballistic missile base trajectory developed in Chapter II. The α - β - γ filter simulation results are presented and its tracking accuracy is analyzed.

A. ALPHA-BETA-GAMMA TRACKER

The system equations for the α - β - γ tracker are the standard tracking equations as presented previously in Chapter III,

$$x_{k+1} = F_k x_k + \omega_k \quad (4.1)$$

$$z_k = H_k x_k + v_k \quad (4.2)$$

where x_k is the missile state vector,

$$x_k = \begin{bmatrix} x - \text{position} \\ x - \text{velocity} \\ x - \text{acceleration} \\ y - \text{position} \\ y - \text{velocity} \\ y - \text{acceleration} \\ z - \text{position} \\ z - \text{velocity} \\ z - \text{acceleration} \end{bmatrix} = \begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \\ z \\ v_z \\ a_z \end{bmatrix} \quad (4.3)$$

and F_k is the known state transition matrix,

$$F_k = \begin{bmatrix} 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

The noise processes ω_k and v_k are the plant noise and measurement noise respectively.

In the α - β - γ tracker, the sensor observes the missile positions in nonlinear range, bearing and elevation measurements. The covariance matrices are not used in this type of filter; consequently, the matrix of partial derivatives (as used in the EKF) is not required. In the

α - β - γ algorithm, the measurements observed by the sensor are simply converted from radar measurements to cartesian coordinates using the following transformation,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \text{range} \times \cos(\text{bearing}) \times \cos(\text{elevation}) \\ \text{range} \times \cos(\text{elevation}) \times \sin(\text{bearing}) \\ \text{range} \times \sin(\text{elevation}) \end{bmatrix} \quad (4.5)$$

and thus H_k , the observation matrix, is simply a constant matrix,

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (4.6)$$

The α - β - γ tracker, as presented in *Multiple-Target Tracking with Radar Applications* [Ref. 7], is comprised of *prediction* and *correction* equations. These equations are as follows:

Prediction:

$$\hat{x}_{k+1} = F_k x_k \quad (4.7)$$

Correction:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \begin{bmatrix} \alpha & 0 & 0 \\ \frac{\beta}{\eta\Delta} & 0 & 0 \\ \frac{\gamma}{(\eta\Delta)^2} & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & \frac{\beta}{\eta\Delta} & 0 \\ 0 & \frac{\gamma}{(\eta\Delta)^2} & 0 \\ 0 & 0 & \alpha \\ 0 & 0 & \frac{\beta}{\eta\Delta} \\ 0 & 0 & \frac{\gamma}{(\eta\Delta)^2} \end{bmatrix} [\tilde{z}_{k+1}] \quad (4.8)$$

where the residual vector, $[\tilde{z}_{k+1}]$, is defined as,

$$\tilde{z}_{k+1} = z_{k+1} - H_{k+1}\hat{x}_{k+1|k} \quad (4.9)$$

The variable η is normally defined to be unity, but in the case when missing observations occur, its value may be taken as the number of scans since the last measurement [Ref. 7].

A large value for η indicates the measurement is discounted. The combined set of *prediction* and *correction* equations along with the constant gain matrix comprises the α - β - γ filter.

The α - β - γ tracker hypothesizes constant missile acceleration; therefore, the gain matrix, as shown in Equation 4.8, is comprised of constant coefficient values for α , β , and γ . Decreasing the coefficient values leads to a less responsive filter. Conversely, increasing the coefficient values leads to better performance for dynamic inputs such as target maneuvers. The relationships between the coefficient values of the gain matrix, as

presented in *Multiple-Target Tracking with Radar Applications* [Ref. 7], are derived to obtain a compromise between noise reduction and maneuver-following capability. The first coefficient value, α , satisfies the relationship

$$0 < \alpha \leq 0.6 \quad (4.10)$$

where a large value of α results in better tracking during target maneuvers. A large value of α puts more emphasis on the measured position rather than the estimated target position in the correction step of the filter. The relationships for β and γ are defined as,

$$\beta = 2(2 - \alpha) - 4\sqrt{1 - \alpha} \quad (4.11)$$

$$\gamma = \frac{\beta^2}{2\alpha} \quad (4.12)$$

The choice of gains for a constant-coefficient filter must reflect an overall compromise between noise and dynamic (maneuver) performance.

B. SIMULATION RESULTS

The α - β - γ tracking algorithm is developed using the α - β - γ equations and is then implemented on the ballistic missile base trajectory with added measurement noise. The results are obtained by running the algorithm in MATLAB[®] and by plotting the average trajectories over 10 simulation runs with $\Delta = 0.1$ seconds, with $\alpha = 0.6$, and with β and γ satisfying the α - β - γ relationships as described in Equations 4.11 and 4.12. The value of α is selected as a large value to see the effect of the filter if a maneuvering target is expected.

Figure 4.1(a) shows the ballistic missile base trajectory during boost phase. As in Chapters II and III, the sensor position is assumed to be a surface platform at coordinates (100 km, 100 km, 0 km), with standard deviations in range, bearing and elevation of 10 meters, 1 degree, and 1 degree respectively. Figure 4.1(b) shows the ballistic missile base trajectory with added measurement noise. The result of the α - β - γ tracking algorithm is shown in Figure 4.1(c), with the filtered trajectory superimposed on the ballistic missile base trajectory. Figures 4.1(d) through (f) show a close-up of the Alpha-Beta-Gamma trajectory at 40 seconds, 60 seconds and 80 seconds respectively.

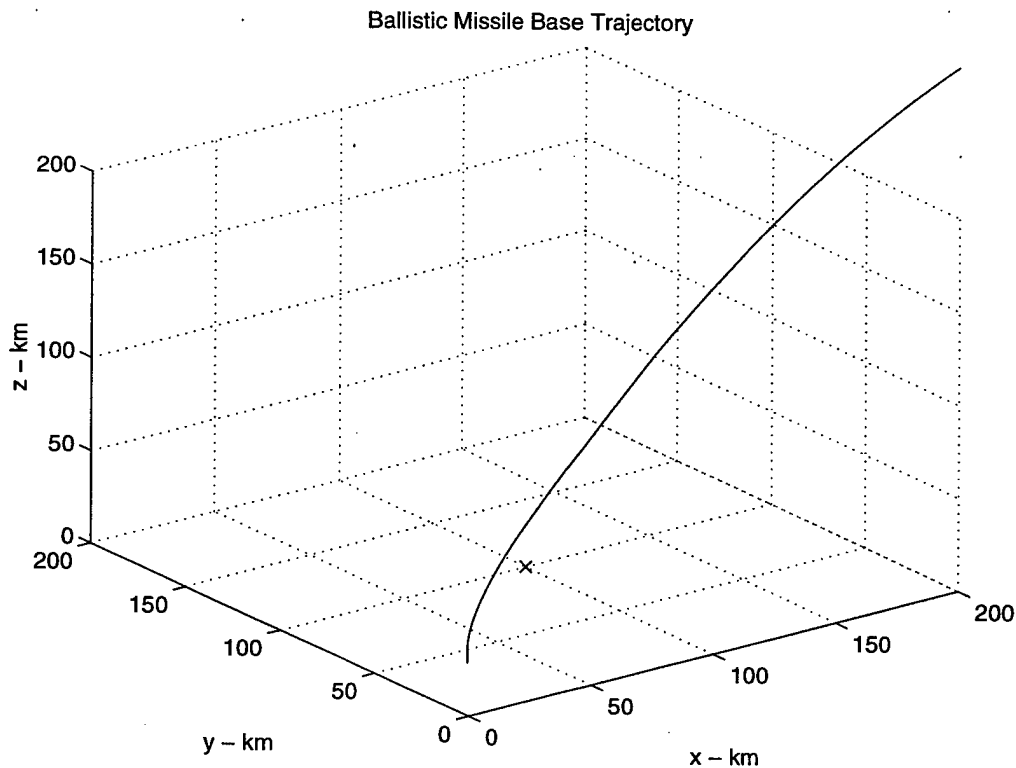


Figure 4.1(a) Ballistic Missile Base Trajectory.

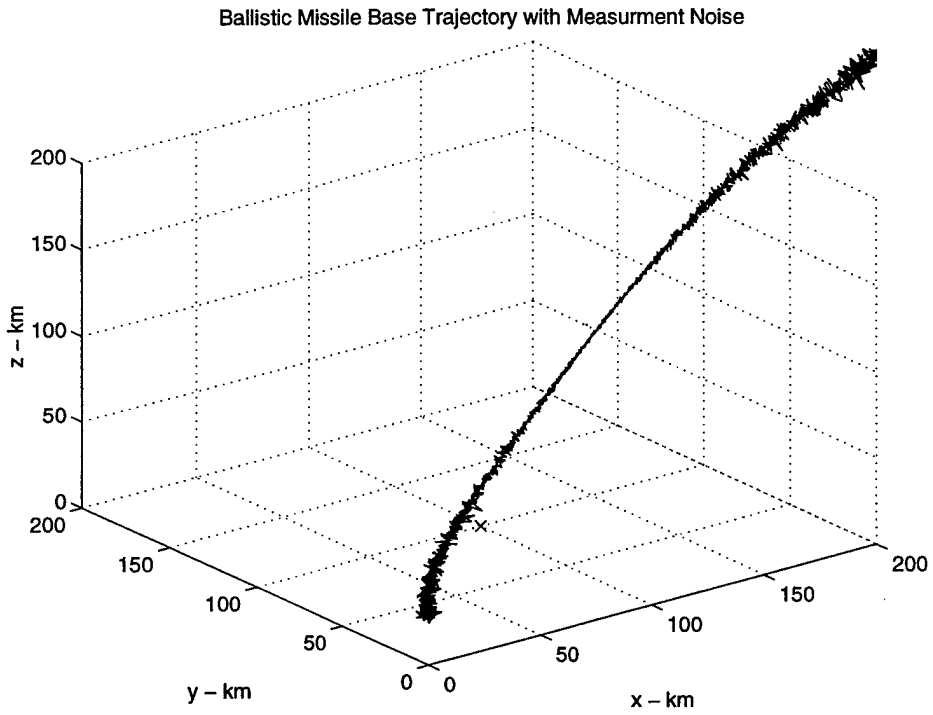


Figure 4.1(b) Ballistic Missile Base Trajectory with Measurement Noise.

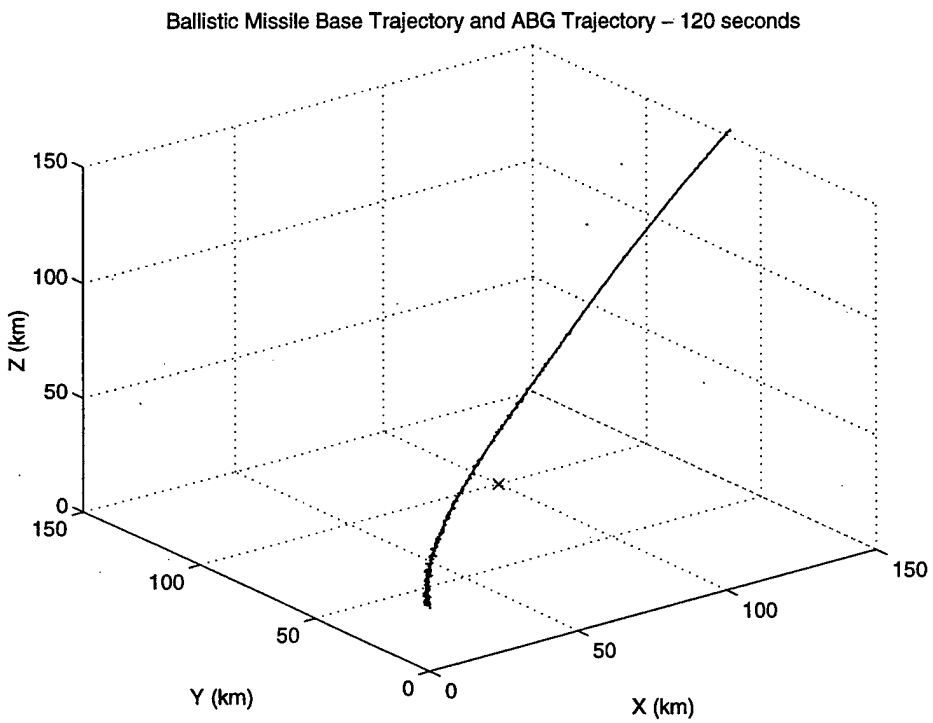


Figure 4.1(c) α - β - γ Trajectory, Initial 120 Seconds (10 runs, $\alpha=0.6$).

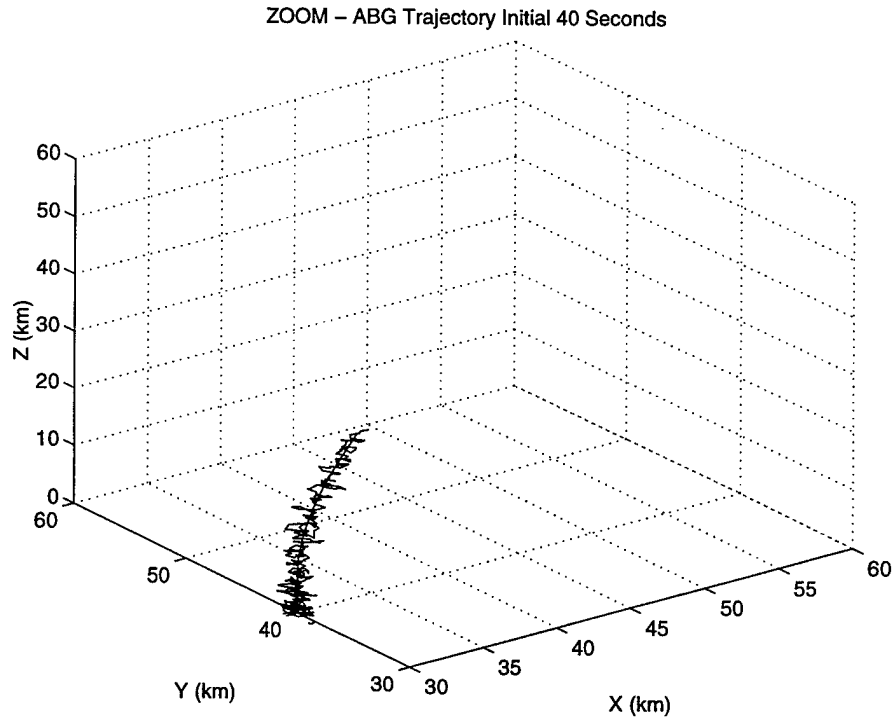


Figure 4.1(d) α - β - γ , Trajectory, Initial 40 Seconds (10 runs, $\alpha=0.6$).

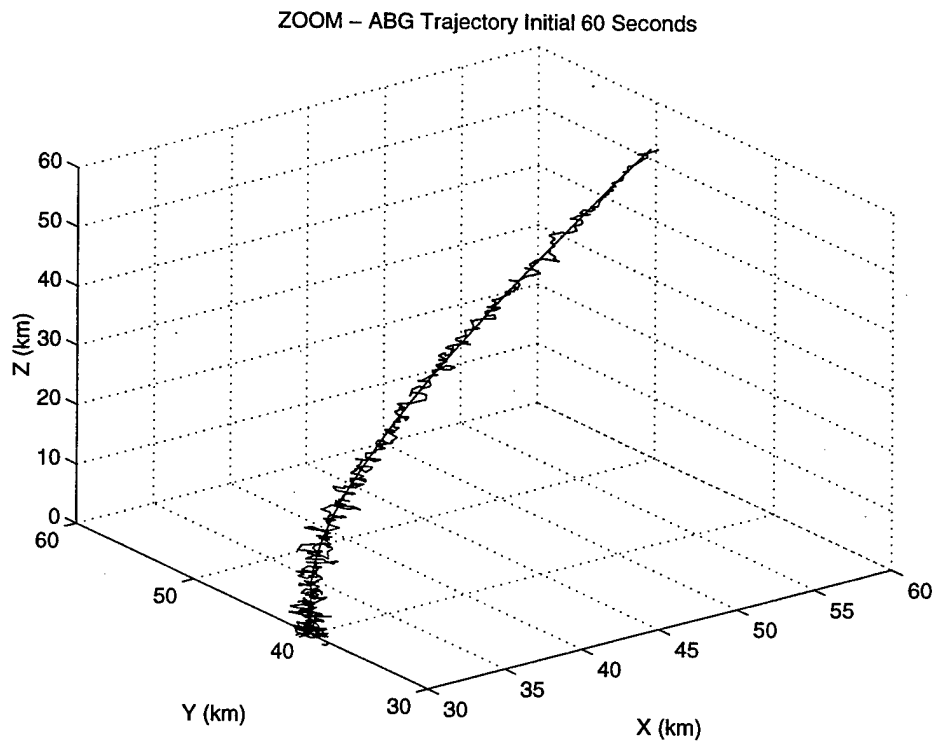


Figure 4.1(e) α - β - γ Trajectory, Initial 60 Seconds (10 runs, $\alpha=0.6$).

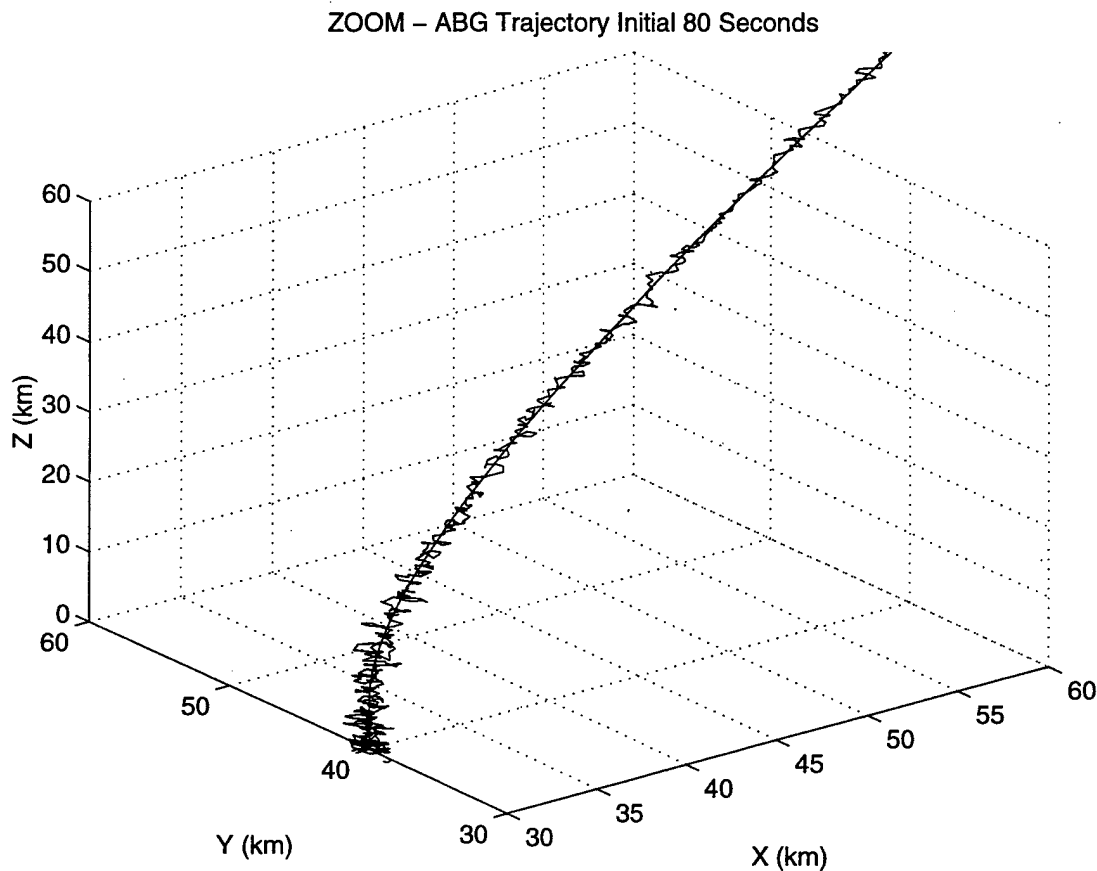


Figure 4.1(f) α - β - γ Trajectory, Initial 80 Seconds (10 runs, $\alpha=0.6$).

The mean distance error in measurements is calculated over 100 simulation runs and is shown in Figure 4.2(a). The upper plot is the mean measurement noise, and the lower plot is the mean distance error using the α - β - γ tracking algorithm. These results indicate that the α - β - γ tracker performs only slightly better than the mean measurement noise observed by the sensor. Additionally, a large transient error is present in the first few seconds of the filter. This is shown in Figure 4.2(a) as a large spike, peaking to approximately 6700 meters. A close-up of the mean distance error, disregarding the initial transient error, is shown in Figure 4.2(b).

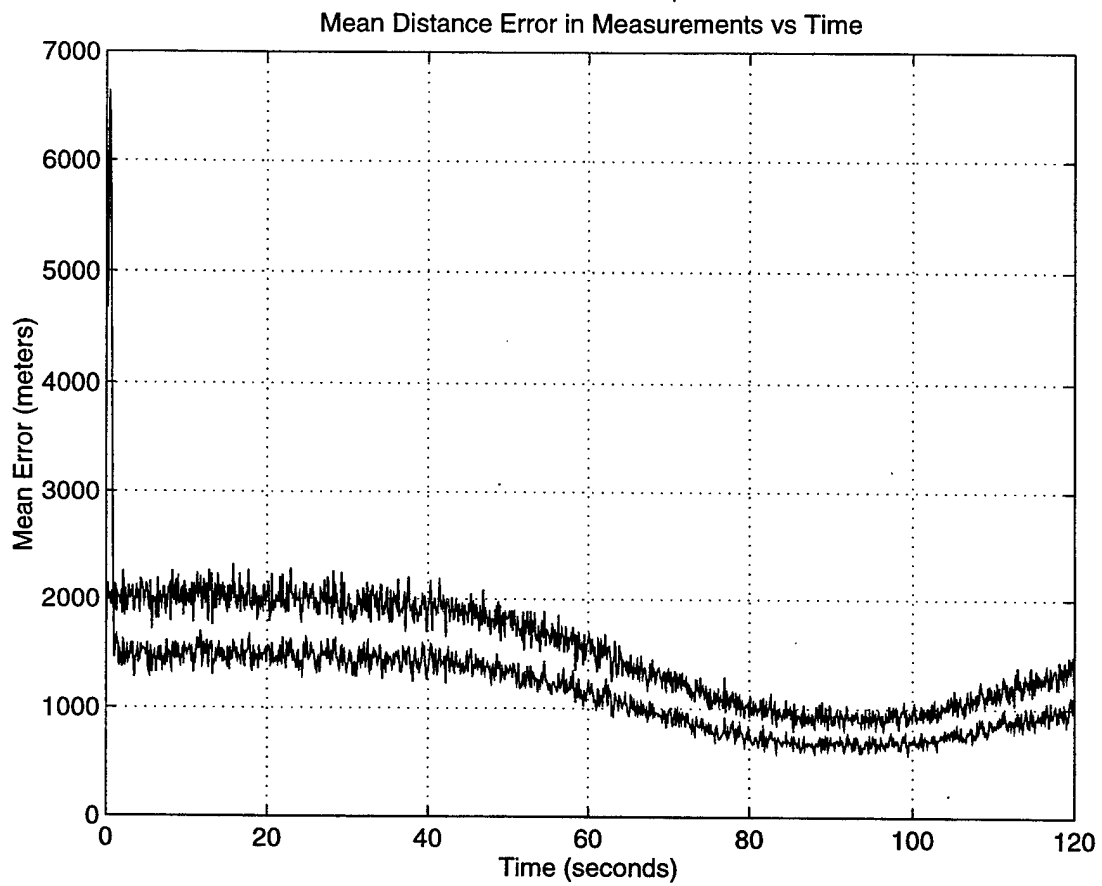


Figure 4.2(a) α - β - γ Mean Distance Error (100 runs, $\alpha=0.6$).

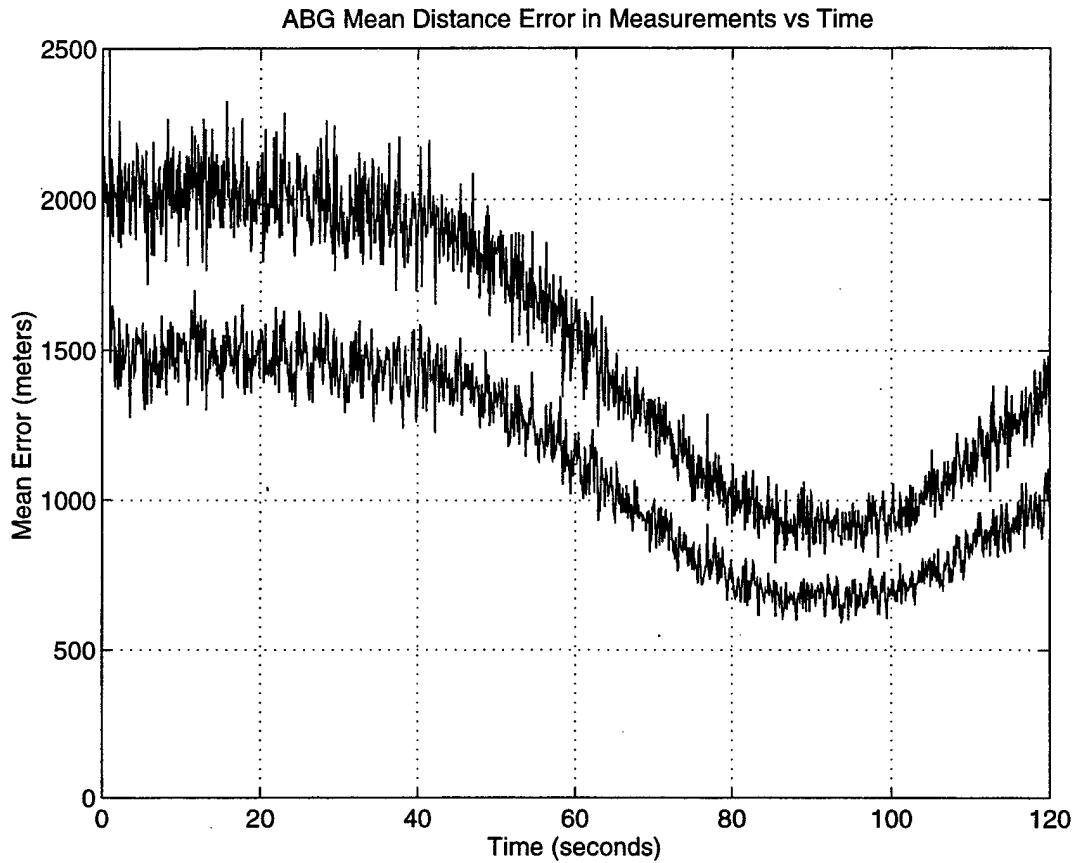


Figure 4.2(b) Close-up, α - β - γ Mean Distance Error (100 runs, $\alpha=0.6$).

Figure 4.3 shows a comparison of the mean distance error plots of the α - β - γ tracker and of the EKF tracking algorithm. The EKF results are shown as a dotted line. Analysis of this graph shows that the EKF tracking algorithm is superior to the α - β - γ tracker throughout the boost phase tracking.

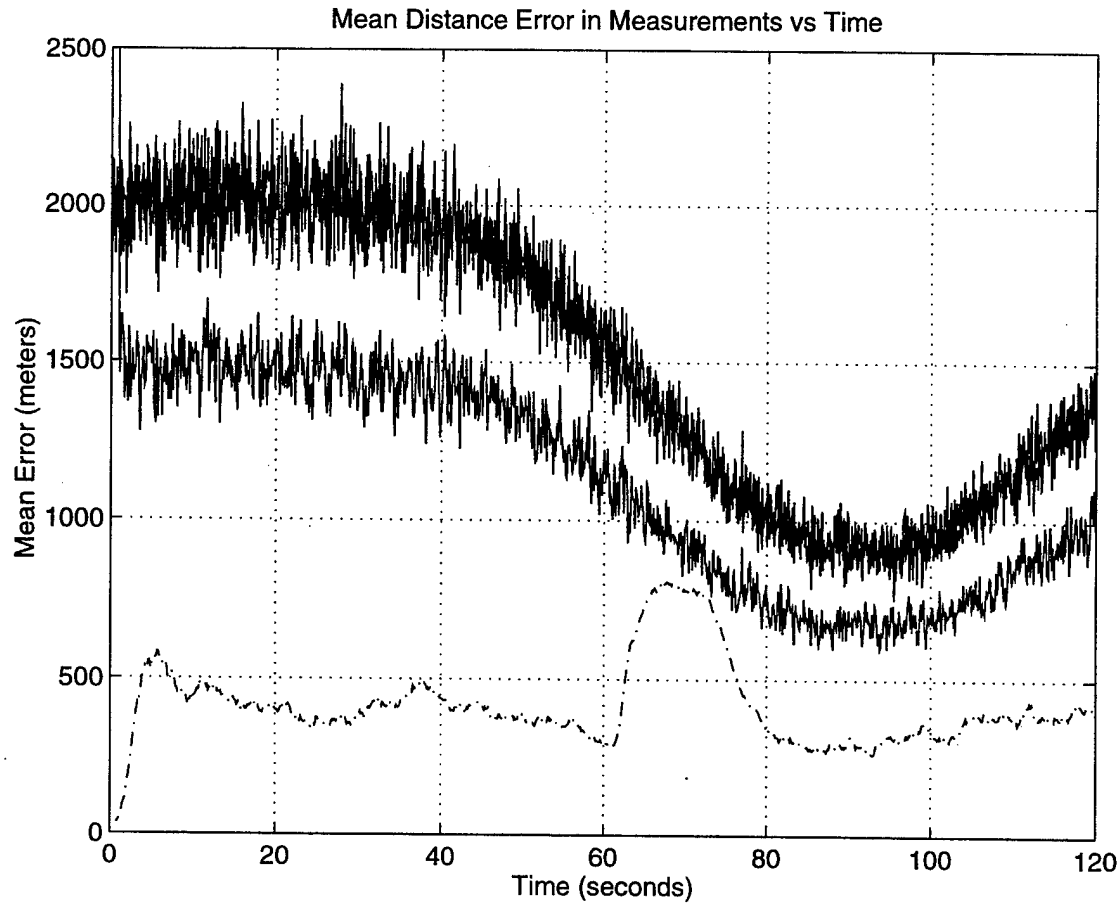


Figure 4.3 Mean Distance Error, α - β - γ Tracker vs. EKF (100 Runs).

In order to see how a different value of α affects the resulting trajectory, an additional simulation was conducted for $\alpha=0.2$, with β and γ satisfying the α - β - γ relationships as described in Equations 4.11 and 4.12. As stated in the previous section, a small value of α leads to a less responsive filter and improved measures of performance for random noise input, whereas a large value of α , leads to better performance for dynamic inputs. Therefore, in this simulation, we expect to see better performance of the filter with $\alpha=0.2$ since random noise is added to the ballistic missile base trajectory. The

mean distance error in measurements was calculated over 100 simulation runs, and is shown in Figure 4.4. As in the previous example, a large transient error is present and is shown in Figure 4.4(a); however, it is noted that the transient error is larger for smaller values of α . With $\alpha=0.2$, the error peaks to approximately 140 kilometers as compared to a transient error of 6700 meters when $\alpha=0.6$. Figure 4.4(b) shows a close up of the mean distance error disregarding the initial transient error. As expected, the mean distance error (the lower plot) is approximately 50 percent of the mean measurement noise (the upper plot). This is significantly lower than the mean distance error for $\alpha=0.6$, as shown

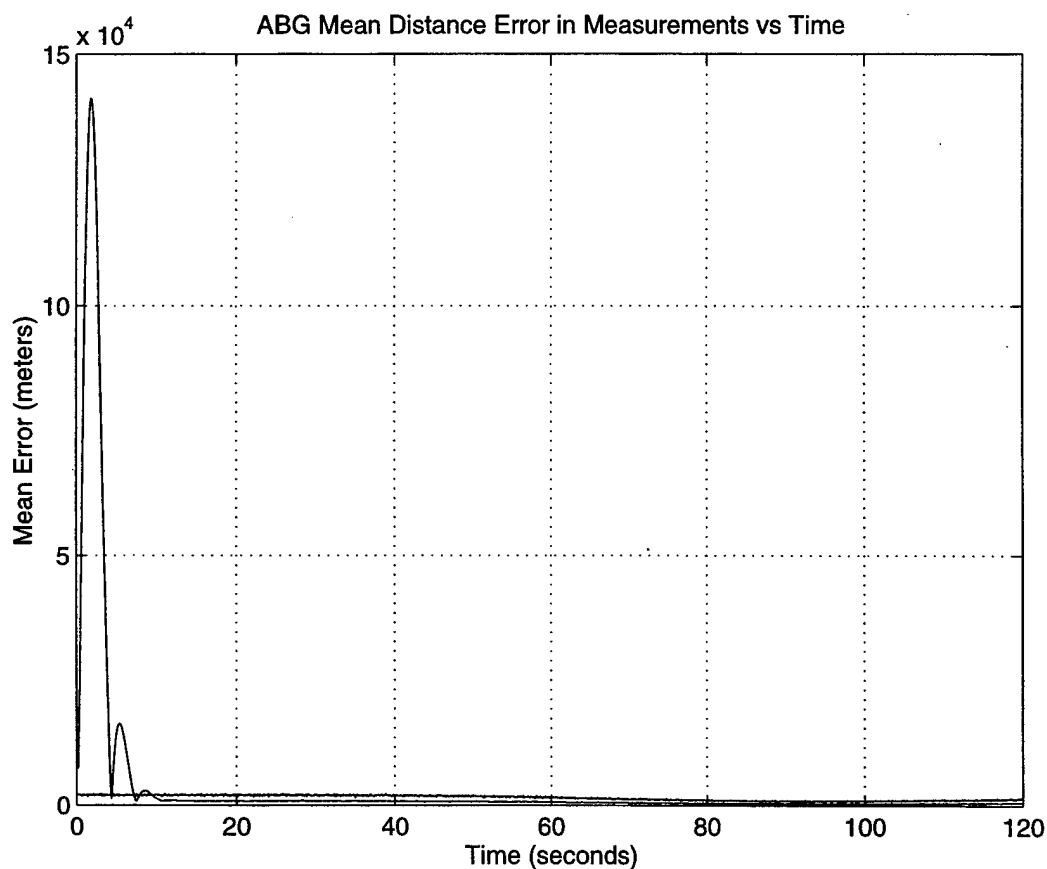


Figure 4.4(a) α - β - γ Mean Distance Error (100 runs, $\alpha=0.2$).

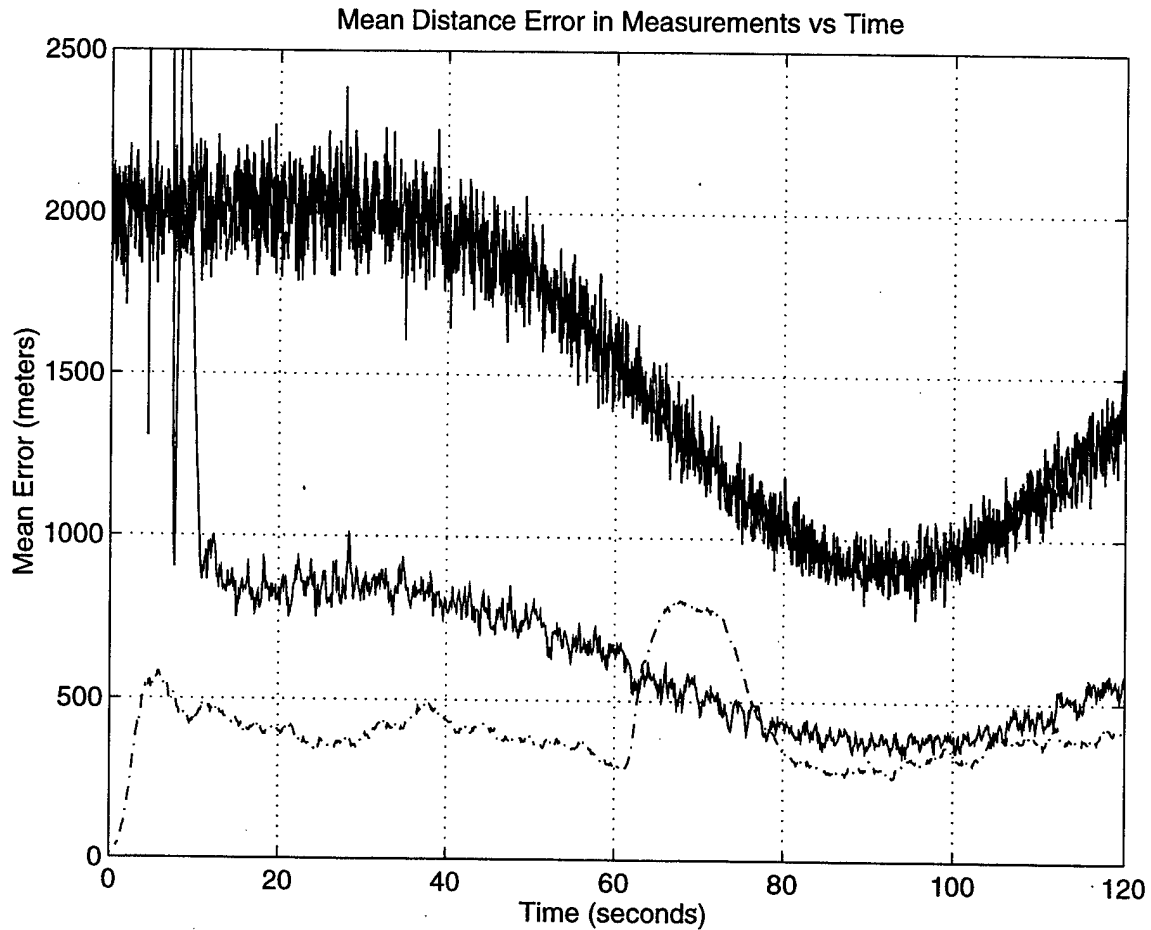


Figure 4.4(b) Close-up, α - β - γ Mean Distance Error (100 runs, $\alpha=0.2$).

in Figure 4.2(b), where the mean distance error is approximately 75 percent of the mean measurement noise. Figure 4.4(b) also shows a comparison of the mean distance error plots of the α - β - γ tracker (with $\alpha=0.2$) and the EKF tracking algorithm. The EKF results are shown as a dotted line. Although the α - β - γ tracker (with $\alpha=0.2$) performs better than the EKF in the areas between 62 and 78 seconds, the EKF is the better overall filter due to the large transient error present in the α - β - γ tracker. In missile tracking, a very large initial transient error is not acceptable, and thus a large value of α ($\alpha=0.6$) is used

throughout the remainder of this study. The MATLAB source code for the α - β - γ tracker is provided in Appendix C.

In the next chapter, the tracking accuracy of one final tracking algorithm is analyzed. Chapter V investigates the missile tracking accuracy of a multiple model system using the Interacting Multiple Model (IMM) algorithm. In this algorithm, an accelerating model and a ballistic model are developed using the EKF equations as presented in Chapter III. These two models are combined in the IMM filter to produce a combined estimate. Simulation results of the IMM are presented, and the tracking accuracy is analyzed.

V. INTERACTING MULTIPLE MODEL ALGORITHM

The Interacting Multiple Model (IMM) tracking algorithm, as outlined in *Multitarget-Multisensor Tracking: Principles and Techniques* [Ref. 10], is a hybrid filter system comprised of a finite number of system models. This multiple model approach provides a versatile tool for adaptive state estimation in systems whose behavior pattern changes with time [Ref. 10]. A ballistic missile encounters two distinct behavior patterns along its trajectory. Initially, the missile experiences large accelerations while the rocket's motor burns. Then, after the motor burns out, the missile enters a purely ballistic state for the remainder of its trajectory. Therefore, in this study, two system models are developed for use in the IMM algorithm: an accelerating model and a ballistic model. State and covariance estimates are calculated and maintained for each model (or mode) and then mixed via a Markov state transition probability matrix. The end result is an overall state and covariance matrix that provides a mode conditioned combination of the latest state estimates and covariances. The details of the IMM algorithm are presented in the following section. As in the previous chapters, the algorithm is implemented on the ballistic missile base trajectory with added measurement noise. Simulation results are presented and the tracking accuracy is analyzed.

A. IMM ALGORITHM

The theatre ballistic missile is assumed to be operating in one of two distinct modes: accelerating (a third order, constant acceleration model) or ballistic (a second

order, constant velocity model). System (or plant) noise accounts for small variations from these assumptions in each model. In the IMM algorithm, each model requires its own EKF system equations. The algorithm consists of operating these two EKF models in parallel, with an interaction between the two filters resulting in the mixing of the estimates. The two models of target motion in this study are defined by the following system and measurement equations. (Note that the superscript in these equations are for notation purposes only, and it indicates the model number of the equation, not an exponential factor.)

Model 1 - Accelerating model:

$$x_{k+1}^1 = F^1 x_k^1 + G^1 + \omega_k^1 \quad (5.1)$$

$$z_k^1 = H^1 x_k^1 + v_k^1 \quad (5.2)$$

where x_k^1 is the missile state vector for the accelerating model,

$$x_k^1 = \begin{bmatrix} \text{x - position} \\ \text{x - velocity} \\ \text{x - acceleration} \\ \text{y - position} \\ \text{y - velocity} \\ \text{y - acceleration} \\ \text{z - position} \\ \text{z - velocity} \\ \text{z - acceleration} \end{bmatrix} = \begin{bmatrix} \text{x} \\ \text{v}_x \\ \text{a}_x \\ \text{y} \\ \text{v}_y \\ \text{a}_y \\ \text{z} \\ \text{v}_z \\ \text{a}_z \end{bmatrix} \quad (5.3)$$

F^1 is the state transition matrix,

$$F^1 = \begin{bmatrix} 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta & \frac{\Delta^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.4)$$

G^1 is the gravity matrix, which accounts for the force of gravity in the z direction

with $g = 9.8 \frac{\text{m}}{\text{s}^2}$,

$$G^1 = -g \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{\Delta^2}{2} \\ \Delta \\ 0 \end{bmatrix} \quad (5.5)$$

H^1 is the matrix of partials in which the missile positions are observed in range, bearing and elevation (nonlinear measurements),

$$H^1 = \begin{bmatrix} \frac{x}{\sqrt{x^2 + y^2 + z^2}} & 0 & 0 & \frac{y}{\sqrt{x^2 + y^2 + z^2}} & 0 & 0 & \frac{z}{\sqrt{x^2 + y^2 + z^2}} & 0 & 0 \\ \frac{-y}{x^2 + y^2} & 0 & 0 & \frac{x}{x^2 + y^2} & 0 & 0 & 0 & 0 & 0 \\ \frac{-xz}{\sqrt{x^2 + y^2}(x^2 + y^2 + z^2)} & 0 & 0 & \frac{-yz}{\sqrt{x^2 + y^2}(x^2 + y^2 + z^2)} & 0 & 0 & \frac{\sqrt{x^2 + y^2}}{x^2 + y^2 + z^2} & 0 & 0 \end{bmatrix} \quad (5.6)$$

and ω_k^1 and v_k^1 are the plant noise (with covariance Q_k^1) and measurement noise (with covariance R_k^1) respectively.

Model 2 - Ballistic model:

$$x_k^2 = F^2 x_k^2 + G^2 + \omega_k^2 \quad (5.7)$$

$$z_k^2 = H^2 x_k^2 + v_k^2 \quad (5.8)$$

where x_k^2 is the missile state vector for the ballistic model,

$$x_k^2 = \begin{bmatrix} x - \text{position} \\ x - \text{velocity} \\ y - \text{position} \\ y - \text{velocity} \\ z - \text{position} \\ z - \text{velocity} \end{bmatrix} = \begin{bmatrix} x \\ v_x \\ y \\ v_y \\ z \\ v_z \end{bmatrix} \quad (5.9)$$

F^2 is the state transition matrix,

$$F^2 = \begin{bmatrix} 1 & \Delta & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.10)$$

G^2 is the gravity matrix,

$$G^2 = -g \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{\Delta^2}{2} \\ \Delta \end{bmatrix} \quad (5.11)$$

H^2 is the observation matrix in which range, bearing and elevation measurements are observed,

$$H^2 = \begin{bmatrix} \frac{x}{\sqrt{x^2 + y^2 + z^2}} & 0 & \frac{y}{\sqrt{x^2 + y^2 + z^2}} & 0 & \frac{z}{\sqrt{x^2 + y^2 + z^2}} & 0 \\ \frac{-y}{x^2 + y^2} & 0 & \frac{x}{x^2 + y^2} & 0 & 0 & 0 \\ \frac{-xz}{\sqrt{x^2 + y^2}(x^2 + y^2 + z^2)} & 0 & \frac{-yz}{\sqrt{x^2 + y^2}(x^2 + y^2 + z^2)} & 0 & \frac{\sqrt{x^2 + y^2}}{(x^2 + y^2 + z^2)} & 0 \end{bmatrix} \quad (5.12)$$

and ω_k^2 and v_k^2 are the plant noise (with covariance Q_k^2) and measurement noise (with covariance R_k^2) respectively.

EKF tracking algorithms (as presented in Chapter III) are developed using these two system models. The two models are run in parallel, and EKF estimates are developed for each model. The state and covariance estimates of each system model are then mixed within the IMM filtering process.

The IMM filtering process for the above two model system is comprised of the following series of computations [Ref. 9, 11]:

STEP 1. Model 1 - Accelerating model:

- A. Calculate the mixing probabilities.
- B. Mix conditions.
- C. Perform the prediction.
- D. Update the measurement.
- E. Score the association.

STEP 2. Model 2 - Ballistic model:

- A. Calculate the mixing probabilities.
- B. Mix conditions.
- C. Perform the prediction.
- D. Update the measurement.
- E. Score the association.

STEP 3. Update the modal likelihood vector.

STEP 4. Produce combined state and covariance estimates.

Each of these steps will be presented in detail; however, to begin our discussion, the *Markov transition matrix* and the *modal likelihood vector* must first be presented.

The IMM algorithm utilizes the Markov transition matrix to specify the changes between the two interacting models. This matrix determines the probabilities of changing state between the measurement times and is represented by ρ , where

$$\rho_{ij} = \rho_{i \rightarrow j} = \text{Pr ob} \left[x_{k+1}^j \middle| x_k^i \right] \quad (5.13)$$

The assumption is that the system jumps between models, with the jumps following a Markov chain transition model. The Markov chain transition probabilities are generally chosen heuristically. In this study, the Markov transition matrix for the two model system is defined as,

$$\begin{bmatrix} \rho_{ij} \end{bmatrix} = \begin{bmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{bmatrix} = \begin{bmatrix} \text{prob}[\text{accel}|\text{accel}] & \text{prob}[\text{ball}|\text{accel}] \\ \text{prob}[\text{accel}|\text{ball}] & \text{prob}[\text{ball}|\text{ball}] \end{bmatrix} \quad (5.14)$$

where

- ρ_{11} is the probability that the missile is accelerating at time, t_{k+1} , if it was accelerating at time, t_k .
- ρ_{12} is the probability that the missile is ballistic at time, t_{k+1} , if it was accelerating at time, t_k .
- ρ_{21} is the probability that the missile is accelerating at time, t_{k+1} , if it was ballistic at time, t_k .
- ρ_{22} is the probability that the missile is ballistic at time, t_{k+1} , if it was ballistic at time, t_k .

Applying the above definition of the Markov transition matrix to missile tracking, this matrix can be further simplified. For example, element ρ_{21} is the probability that the missile is accelerating at the next measurement time, given that it is currently ballistic. This certainly can never happen, as the missile enters a purely ballistic trajectory after the rocket motor burns out. Therefore, this element has a zero probability of occurring. Furthermore, element ρ_{22} is the probability that the missile is ballistic at the next measurement time, given that it is currently ballistic. Using the same explanation, this should always be true. Therefore, in this study the Markov transition matrix is simplified to

$$\begin{bmatrix} \rho_{ij} \end{bmatrix} = \begin{bmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{bmatrix} = \begin{bmatrix} \rho_{11} & \rho_{12} \\ 0 & 1 \end{bmatrix} \quad (5.15)$$

It should be noted that according to the *law of total probability*, the rows of the Markov transition matrix sum to one.

Elements ρ_{11} and ρ_{12} are important in the process of switching between models. The element ρ_{11} is the probability that the missile continues accelerating at the next measurement time, and ρ_{12} is the probability that the missile switches from the accelerating model to the ballistic model at the next measurement time. Since our first measurement will occur during boost phase, the value of ρ_{11} is initially set to one and ρ_{12} is initially set to zero. However, as we continue to track the missile, the value of ρ_{12} will increase since there is an increasing probability that the missile will switch to the ballistic model at the next measurement time. In this study, the switching process is modeled using a sigmoid function to switch element ρ_{11} from a value of 1.0 to 0.5. This sigmoid

switching process is designed as a function of altitude since we expect the tracking algorithm to anticipate the switch between missile models after the missile reaches a predetermined minimum altitude. In our ballistic missile base trajectory, it is known that the booster cut off in the simulated missile occurs at an altitude of approximately 60 kilometers. Therefore, in this study the switching process in the IMM algorithm is set to anticipate the change in models after the missile reaches an altitude of 50 kilometers. The sigmoid switching function is designed as follows,

$$\rho_{11}(z) = -0.5 \left(\frac{1}{1 + e^{-0.0005(z-50\text{km})}} - (1 + 1.0) \right) \quad (5.16)$$

The element ρ_{12} is then determined by $\rho_{12} = 1 - \rho_{11}$. Since the time interval between measurements in our ballistic missile simulation is only 0.1 seconds, we assign a minimum value of $\rho_{11} = 0.5$. In the case where the time interval between measurements is larger (i.e., 1 second, or 2 seconds), the sigmoid function should be designed to switch ρ_{11} from 1.0 to a smaller value such as 0.1 or 0.2. Simply put, there would be a smaller probability that the missile would continue to accelerate over the larger time interval between measurements.

Along with the Markov transition matrix, the IMM algorithm utilizes the *modal likelihood vector* in the mixing process. The modal likelihood vector, μ_k , maintains the current set of probabilities for each modal state and changes with each update cycle as the missile maneuvers. After the measurement update step in each system model, the modal likelihoods are updated based on a scoring technique, which accounts for the latest

measurement. The modal likelihood of each state is defined as μ_k^i , which is the likelihood of being in state i at time, t_k . For this algorithm, μ^1 represents the probability that the missile is currently accelerating, and μ^2 represents the probability that the missile is currently ballistic. The sum of the probabilities from each modal state is defined to equal one. The *modal likelihood vector* for our two model system is defined as,

$$\mu_k = \begin{bmatrix} \mu_k^1 \\ \mu_k^2 \end{bmatrix} = \begin{bmatrix} \text{probability_the_missile_is_accelerating_at_time_}t_k \\ \text{probability_the_missile_is_ballistic_at_time_}t_k \end{bmatrix} \quad (5.17)$$

The elements of the *modal likelihood vector* and the previously defined *Markov state transition matrix* are used in the first steps of the IMM filtering process. [Ref. 11]

The filtering steps of the IMM algorithm can now be presented. As in the Kalman Filter algorithm, the initial state and covariance estimates for each model are required, where \hat{x}_0^1 and P_0^1 are the initial state and covariance estimates for the accelerating model, and \hat{x}_0^2 and P_0^2 are the initial state and covariance estimates for the ballistic model. Additionally, the *initial* modal likelihood vector is required. Applying the above definition of the modal likelihood vector, the initial modal likelihood vector (evaluated at time t_0) is determined to be,

$$\mu_0 = \begin{bmatrix} \mu_0^1 \\ \mu_0^2 \end{bmatrix} \quad (5.18)$$

The initialization of the modal likelihood vector can be further simplified. In this study we assume that the missile is observed initially during boost phase. Hence, μ_0^1 is initially

set equal to one, and μ_0^2 is set equal to zero. Thus the initial modal likelihood vector is simply,

$$\mu_0 = \begin{bmatrix} \mu_0^1 \\ \mu_0^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (5.19)$$

The first step of the filtering process utilizes the initial state, covariance and modal likelihood estimates to determine a mixed state and covariance estimate for the accelerating model. Similarly, the second step performs the same mixed state and covariance estimates for the ballistic model. Step three updates the modal probabilities μ_k^1 and μ_k^2 utilizing a scoring process. Finally, these updated modal probabilities are used to produce a combined estimate in step four. One cycle of the IMM algorithm consists of the following steps:

STEP 1A. (Model 1) - Calculate the mixing probabilities.

In this algorithm, the mixing is carried out at the beginning of the cycle. The mixing probability, $\mu_{k-1|k-1}^{i|j}$, is defined as the probability that mode M_i was in effect at time t_{k-1} , given that mode M_j is in effect at time t_k . The mixing probabilities for the accelerating model are defined as

$$\mu_{k-1|k-1}^{1|1} = \frac{\rho^{11} \mu_{k-1}^1}{\bar{c}^1} \quad (5.20a)$$

$$\mu_{k-1|k-1}^{2|1} = \frac{\rho^{21} \mu_{k-1}^2}{\bar{c}^1} \quad (5.20b)$$

with the normalizing constant

$$\bar{c}^1 \equiv \rho_{11}\mu_{k-1}^1 + \rho_{21}\mu_{k-1}^2 \quad (5.21)$$

STEP 1B. Mixing.

The mixed initial condition for the accelerating filter is defined as,

$$\hat{x}_{k-1|k-1}^{01} = \hat{x}_{k-1|k-1}^1 \times \mu_{k-1|k-1}^{1|1} + \hat{x}_{k-1|k-1}^2 \times \mu_{k-1|k-1}^{2|1} \quad (5.22)$$

with the corresponding covariance,

$$\begin{aligned} P_{k-1|k-1}^{01} = & \mu_{k-1|k-1}^{1|1} \left[P_{k-1|k-1}^1 + \tilde{x}_{k-1|k-1}^{1|1} \left(\tilde{x}_{k-1|k-1}^{1|1} \right)^T \right] + \\ & \mu_{k-1|k-1}^{2|1} \left[P_{k-1|k-1}^2 + \tilde{x}_{k-1|k-1}^{2|1} \left(\tilde{x}_{k-1|k-1}^{2|1} \right)^T \right] \end{aligned} \quad (5.23)$$

where

$$\tilde{x}_{k-1|k-1}^{i|1} \equiv \hat{x}_{k-1|k-1}^i - \hat{x}_{k-1|k-1}^{01} \quad (5.24)$$

STEP 1C. Perform the prediction.

The state and covariance predictions for the accelerating model are determined by the following equations,

$$\hat{x}_{k|k-1}^1 = F^1 \hat{x}_{k-1|k-1}^{01} \quad (5.25)$$

$$P_{k|k-1}^1 = F^1 P_{k-1|k-1}^{01} (F^1)^T + Q_k^1 \quad (5.26)$$

STEP 1D. Update the measurement.

$$\hat{x}_{k|k}^1 = \hat{x}_{k|k-1}^1 + K_k^1 (z_k - H^1 \hat{x}_{k|k-1}^1) \quad (5.27)$$

$$P_{k|k-1}^1 = (I - K_k^1 H^1) P_{k|k-1}^1 \quad (5.28)$$

where the Kalman gain is defined as,

$$K_k^1 \equiv P_{k|k-1}^1 (H_k^1)^T \left[H_k^1 P_{k|k-1}^1 (H_k^1)^T + R_k \right]^{-1} \quad (5.29)$$

STEP 1E. Score the association. (Based on a likelihood of the observed measurement).

$$\Lambda_k^1 = \frac{e^{\left[-(\bar{z}^1)^T (S_k^1)^{-1} \frac{\bar{z}^1}{2} \right]}}{(2\pi)^{\frac{m}{2}} |S_k^1|^{\frac{1}{2}}} \quad (5.30)$$

where

$$\bar{z}^1 \equiv z_k - H_k^1 \hat{x}_{k|k}^1 \quad (5.31)$$

$$S_k^1 \equiv H_k^1 P_{k|k-1}^1 (H_k^1)^T + R_k \quad (5.32)$$

and m is defined as the number of dimensions observed; thus $m = 3$ since range, bearing and elevation positions are observed.

STEP 2A. (Model 2) - Calculate the mixing probabilities.

The mixing probabilities for the ballistic model are defined as

$$\mu_{k-1|k-1}^{12} = \frac{\rho^{12} \mu_{k-1}^1}{c^2} \quad (5.33a)$$

$$\mu_{k-1|k-1}^{2|2} = \frac{\rho^{22} \mu_{k-1}^2}{\bar{c}^2} \quad (5.33b)$$

with the normalizing constant

$$\bar{c}^2 \equiv \rho^{12} \mu_{k-1}^1 + \rho^{22} \mu_{k-1}^2 \quad (5.34)$$

STEP 2B. Mixing.

The mixed initial condition for the ballistic filter is defined as,

$$\hat{x}_{k-1|k-1}^{02} = \hat{x}_{k-1|k-1}^1 \times \mu_{k-1|k-1}^{1|2} + \hat{x}_{k-1|k-1}^2 \times \mu_{k-1|k-1}^{2|2} \quad (5.35)$$

with the corresponding covariance,

$$\begin{aligned} P_{k-1|k-1}^{02} = & \mu_{k-1|k-1}^{1|2} \left[P_{k-1|k-1}^1 + \tilde{x}_{k-1|k-1}^{1|2} \left(\tilde{x}_{k-1|k-1}^{1|2} \right)^T \right] + \\ & \mu_{k-1|k-1}^{2|2} \left[P_{k-1|k-1}^2 + \tilde{x}_{k-1|k-1}^{2|2} \left(\tilde{x}_{k-1|k-1}^{2|2} \right)^T \right] \end{aligned} \quad (5.36)$$

where

$$\tilde{x}_{k-1|k-1}^{i2} \equiv \hat{x}_{k-1|k-1}^i - \hat{x}_{k-1|k-1}^{02} \quad (5.37)$$

STEP 2C. Perform the prediction.

The state and covariance predictions for the ballistic model are determined by the following equations,

$$\hat{x}_{k|k-1}^2 = F^2 \hat{x}_{k-1|k-1}^{02} \quad (5.38)$$

$$P_{k|k-1}^2 = F^2 P_{k|k-1}^{02} (F^2)^T + Q_k^2 \quad (5.39)$$

STEP 2D. Update the measurement.

$$\hat{x}_{k|k}^2 = \hat{x}_{k|k-1}^2 + K_k^2 (z_k - H^2 \hat{x}_{k|k-1}^2) \quad (5.40)$$

$$P_{k|k-1}^2 = (I - K_k^2 H^2) P_{k|k-1}^2 \quad (5.41)$$

where the Kalman gain is defined as,

$$K_k^2 \equiv P_{k|k-1}^2 (H^2)^T [H_k^2 P_{k|k-1}^2 (H^2)^T + R_k]^{-1} \quad (5.42)$$

STEP 2E. Score the association. (Based on a likelihood of the observed measurement).

$$\Lambda_k^2 = \frac{e^{\left[-(\tilde{z}^2)^T (S_k^2)^{-1} \frac{\tilde{z}^2}{2} \right]}}{(2\pi)^{\frac{m}{2}} |S_k^2|^{\frac{1}{2}}} \quad (5.43)$$

where

$$\tilde{z}^2 \equiv z_k - H_k^2 \hat{x}_{k|k}^2 \quad (5.44)$$

$$S_k^2 \equiv H_k^2 P_{k|k-1}^2 (H_k^2)^T + R_k \quad (5.45)$$

STEP 3. Update the modal likelihoods.

$$\mu_k^1 = \Lambda_k^1 \frac{\bar{c}^1}{c} \quad (5.46)$$

$$\mu_k^2 = \Lambda_k^2 \frac{\bar{c}^2}{c} \quad (5.47)$$

where

$$c \equiv \Lambda_k^1 \bar{c}^1 + \Lambda_k^2 \bar{c}^2 \quad (5.48)$$

STEP 4. Produce combined estimates (for display purposes only).

$$\hat{x}_{k|k} = \mu_k^1 \hat{x}_{k|k}^1 + \mu_k^2 \hat{x}_{k|k}^2 \quad (5.49)$$

$$P_{k|k} = \mu_k^1 \left(P_{k|k}^1 + \left[\hat{x}_{k|k}^1 - \hat{x}_{k|k} \right] \left[\hat{x}_{k|k}^1 - \hat{x}_{k|k} \right]^T \right) + \quad (5.50)$$

$$\mu_k^2 \left(P_{k|k}^2 + \left[\hat{x}_{k|k}^2 - \hat{x}_{k|k} \right] \left[\hat{x}_{k|k}^2 - \hat{x}_{k|k} \right]^T \right)$$

An IMM ballistic missile tracking algorithm is developed in MATLAB using the equations defined in steps one through four. The IMM algorithm is then implemented on the position measurements of the ballistic missile base trajectory. Simulation results of the accelerating model, the ballistic model, and the combined IMM algorithm are presented in the following section. The source code for the IMM algorithm is provided in Appendix C.

B. SIMULATION RESULTS

As in the previous chapters, the IMM algorithm is implemented on the noisy position measurements of the ballistic missile simulation. For the purpose of comparison, the IMM tracking algorithm is run in MATLAB, using the same sensor position,

sampling interval, and measurement uncertainties as in the EKF and the α - β - γ tracker.

Figure 5.1(a) shows the ballistic missile base trajectory and Figure 5.1(b) shows the base trajectory with added measurement noise. Figures 5.2(a) and (b) show the results of the EKF algorithm on the accelerating model. As explained in Chapter III, the accelerating model EKF tracks the missile well until the rocket motors cut off (at time 60 seconds), and the missile changes from an accelerating state to a ballistic state. This discontinuity can be seen in Figure 5.2(b), where the mean distance error at 60 seconds rises from 300 meters to a peak of 800 meters. At approximately 70 seconds, the EKF regains track and the mean distance error decreases below 500 meters, and then remains at approximately 400 meters for the duration of the observation period. Figures 5.3(a) and (b) show the results of the EKF algorithm on the ballistic model. Contrary to the accelerating model EKF, this algorithm has significant difficulty tracking the missile in the early stages of its trajectory. The ballistic model EKF is only able to satisfactorily track the missile after it changes to a ballistic state. Figure 5.3(b) shows that the tracking algorithm reaches a peak mean distance error of approximately 10 kilometers. Once the missile is in a ballistic state, the algorithm is able to regain track.

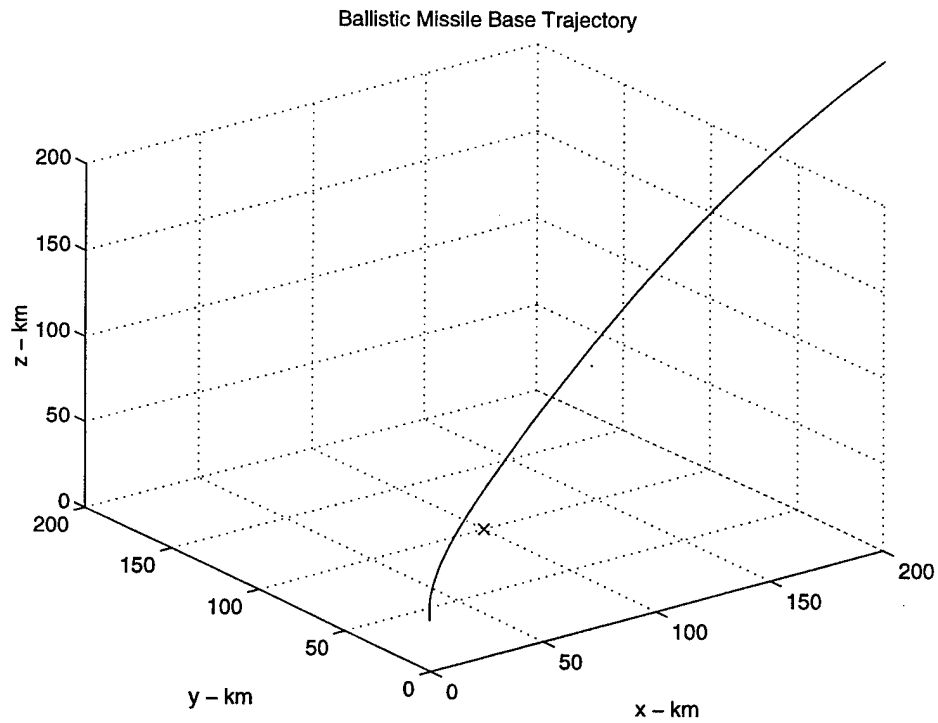


Figure 5.1(a) Ballistic Missile Base Trajectory.

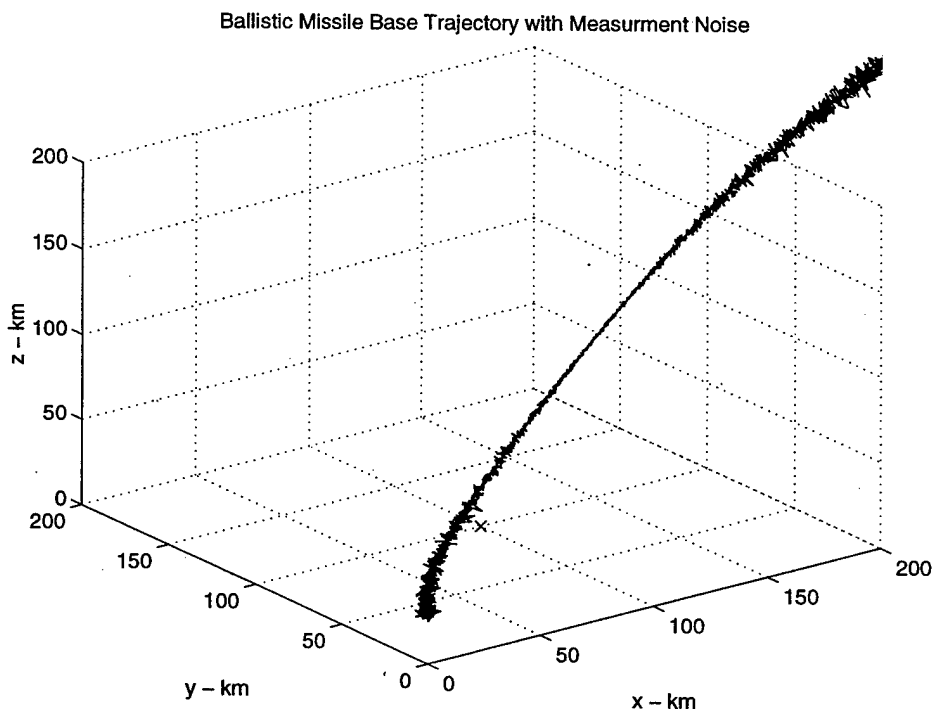


Figure 5.1(b) Ballistic Missile Base Trajectory with Measurement Noise.

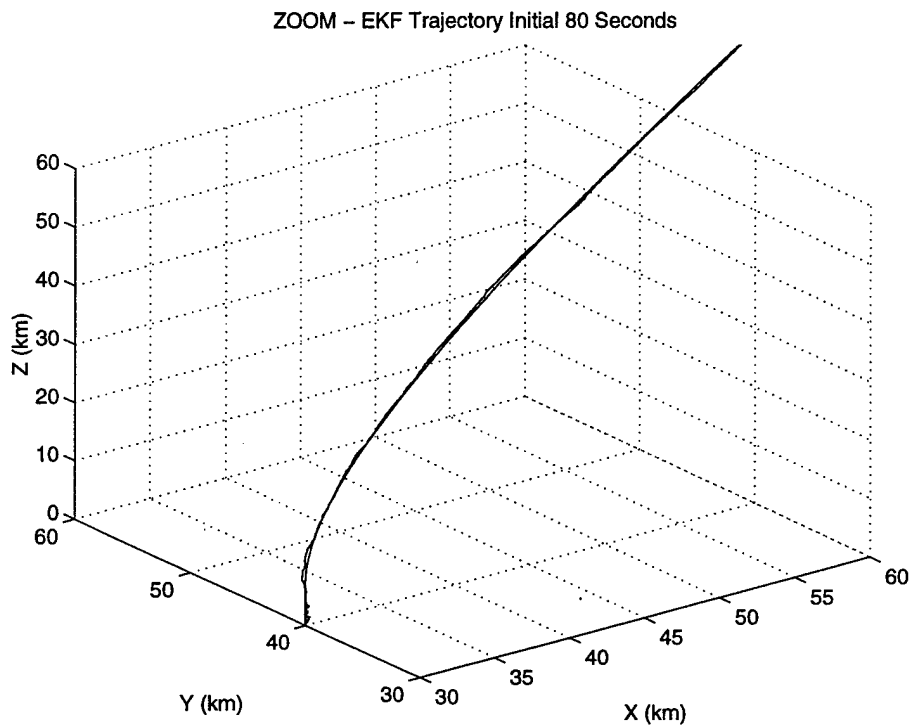


Figure 5.2(a) EKF (Accelerating Model) Trajectory (10 Runs).

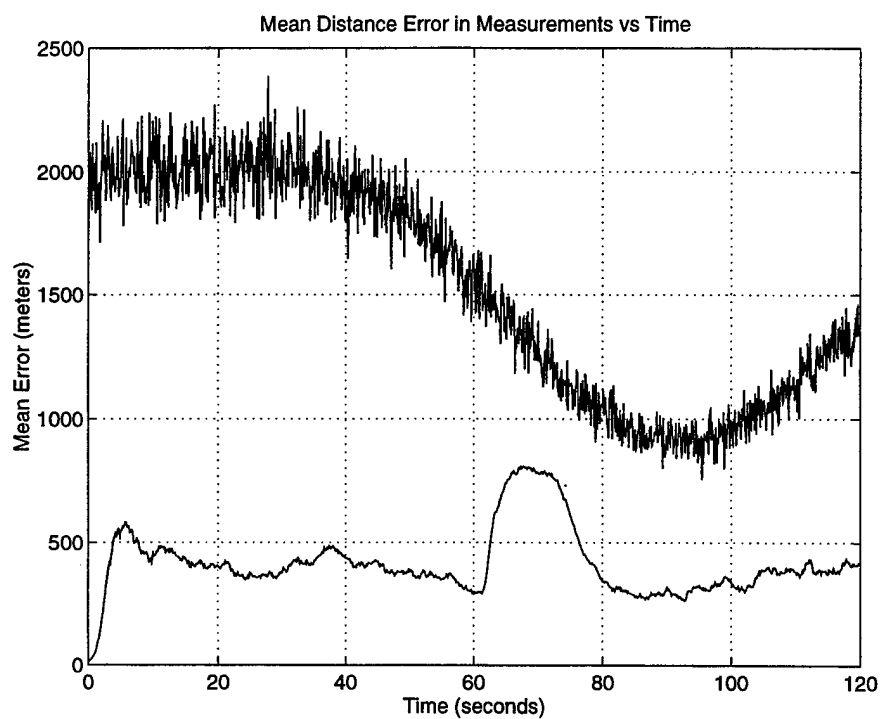


Figure 5.2(b) EKF (Accelerating Model) Mean Distance Error (100 Runs).

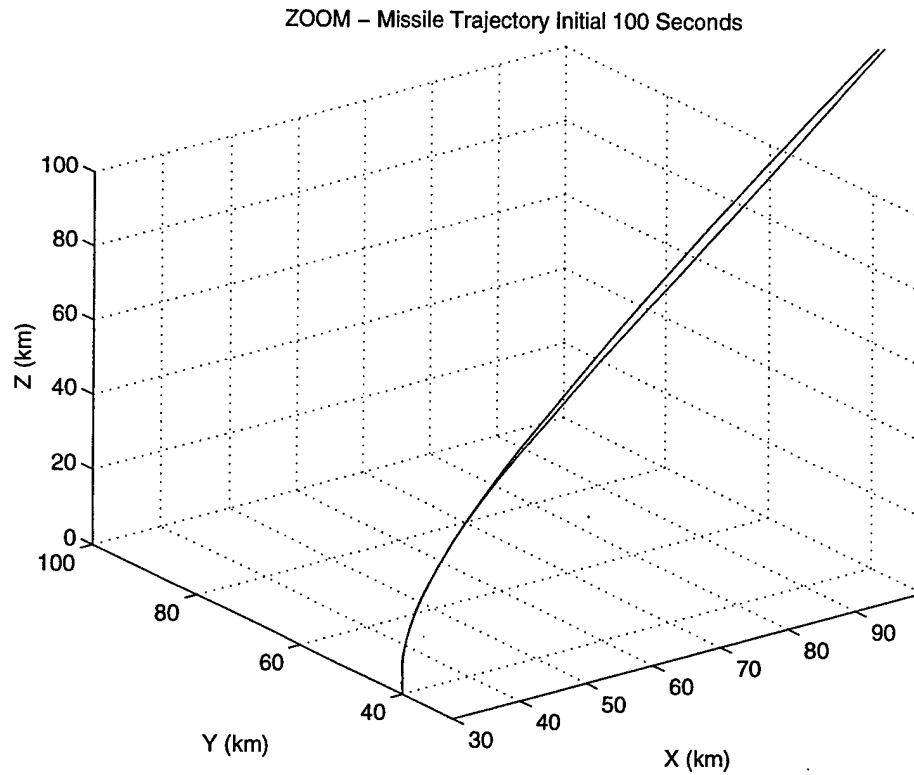


Figure 5.3(a) EKF (Ballistic Model) Trajectory (10 Runs).

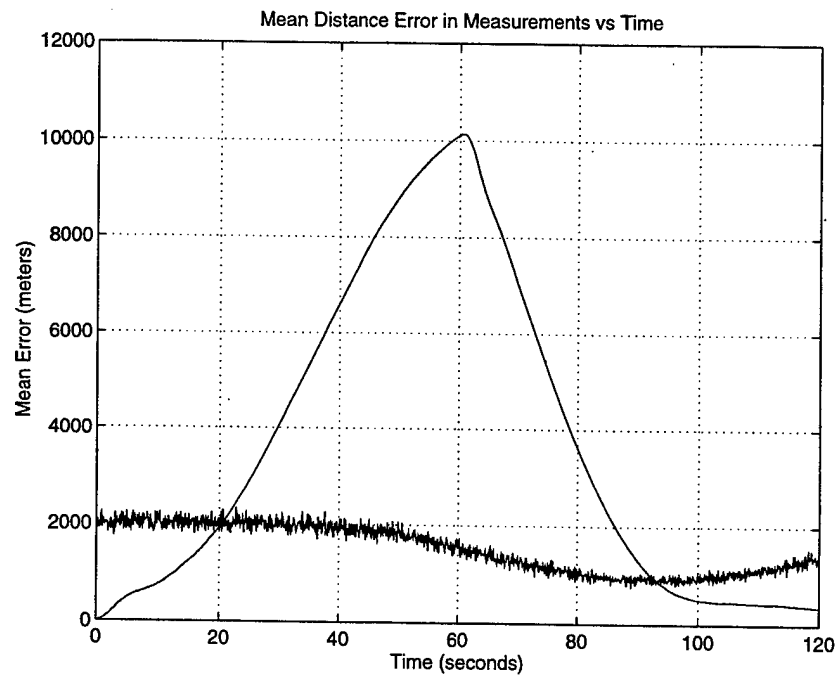


Figure 5.3(b) EKF (Ballistic Model) Mean Distance Error (100 Runs).

The results of the IMM algorithm are shown in Figures 5.4 through 5.6. Figures 5.4(a) through (c) show a close-up of the IMM trajectory at 40 seconds, 60 seconds and 80 seconds respectively. Figure 5.4(d) shows the mean distance error of the IMM algorithm. This graph shows that the IMM algorithm is able to track the missile significantly better than both the accelerating model and the ballistic model. The “problem area” for the accelerating model is the area in which the missile transitions from an accelerating state to a ballistic state. The result, as shown in Figure 5.2(b), is a large rise in the mean distance error that peaks to 800 meters. In the IMM algorithm, this problem area is eliminated, and the IMM algorithm is able to track through the transition area with a mean distance error of approximately 250 meters. The “problem area” for the ballistic model is the initial tracking while the missile is accelerating. This is also resolved, as the IMM algorithm is able to track the missile well in this area with a mean distance error of approximately 500 meters. Figures 5.5(a) and (b) show a comparison of the mean distance error plots for the EKF accelerating model, the EKF ballistic model and the IMM algorithm. Figure 5.5(b) shows a close-up of the comparison. Figure 5.6 shows a comparison of the mean distance error plots for the IMM algorithm and the α - β - γ tracker. Figures 5.5(b) and 5.6 reveal the overall improvement in the tracking capability of the IMM algorithm.

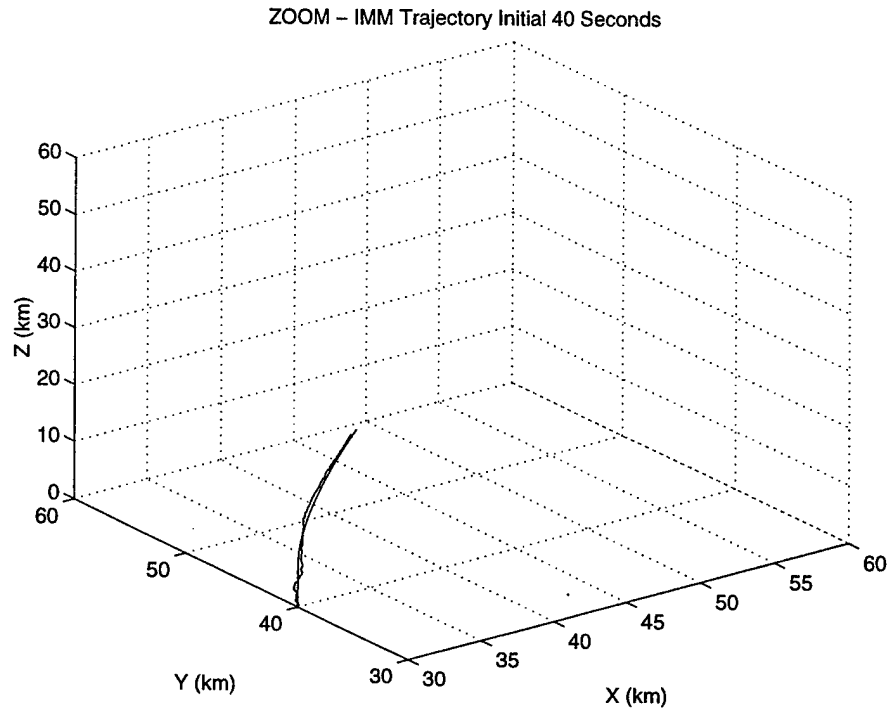


Figure 5.4(a) IMM Trajectory, Initial 40 Seconds (10 Runs).

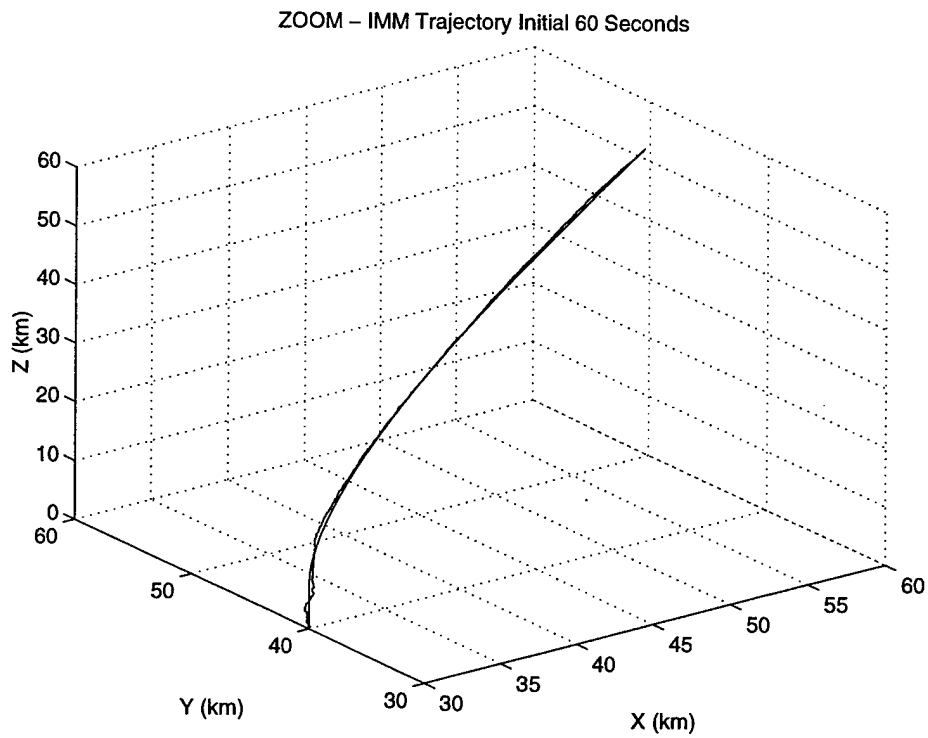


Figure 5.4(b) IMM Trajectory, Initial 60 Seconds (10 Runs).

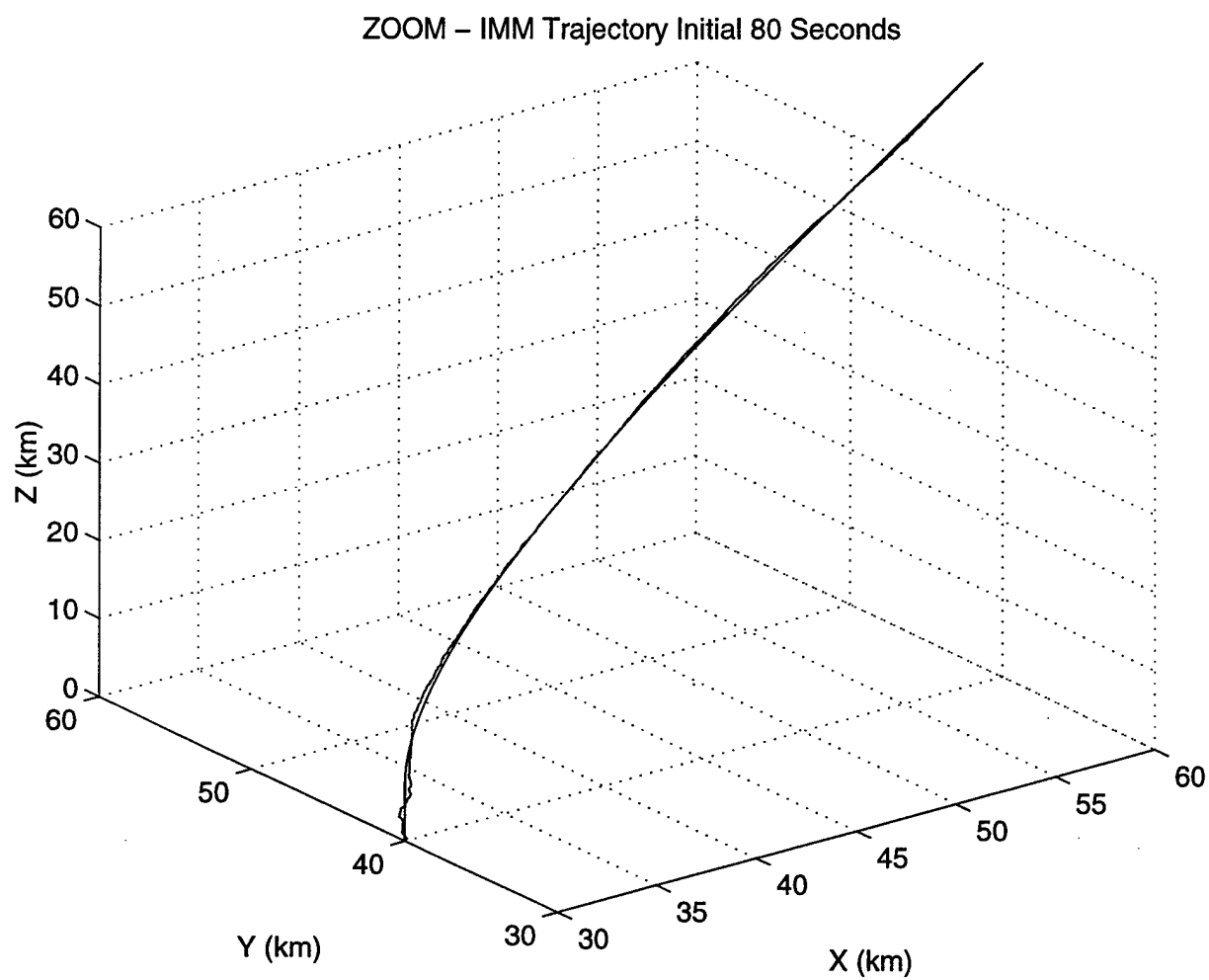


Figure 5.4(c) IMM Trajectory, Initial 80 Seconds (10 Runs).

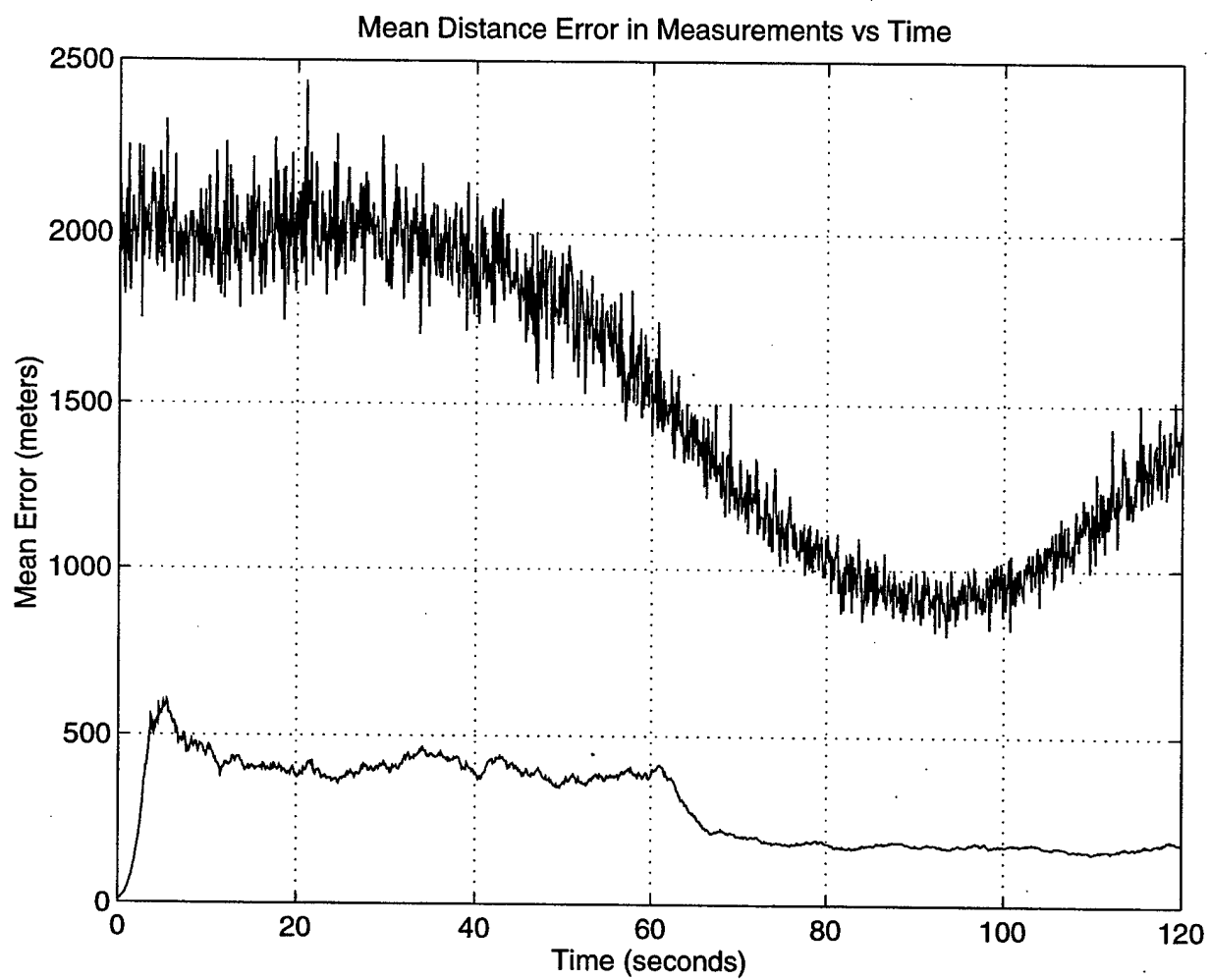


Figure 5.4(d) IMM Mean Distance Error (100 Runs).

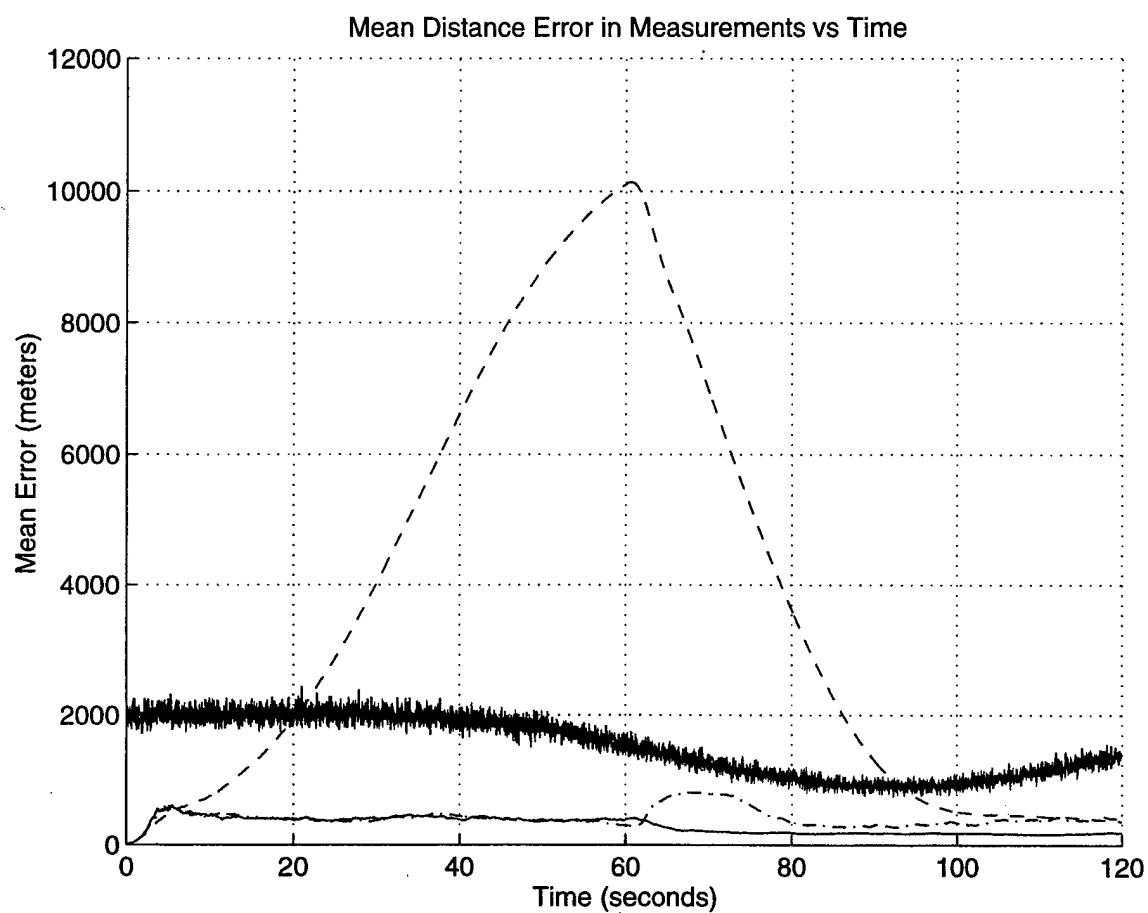


Figure 5.5(a) Comparison of Mean Distance Error (100 Runs).

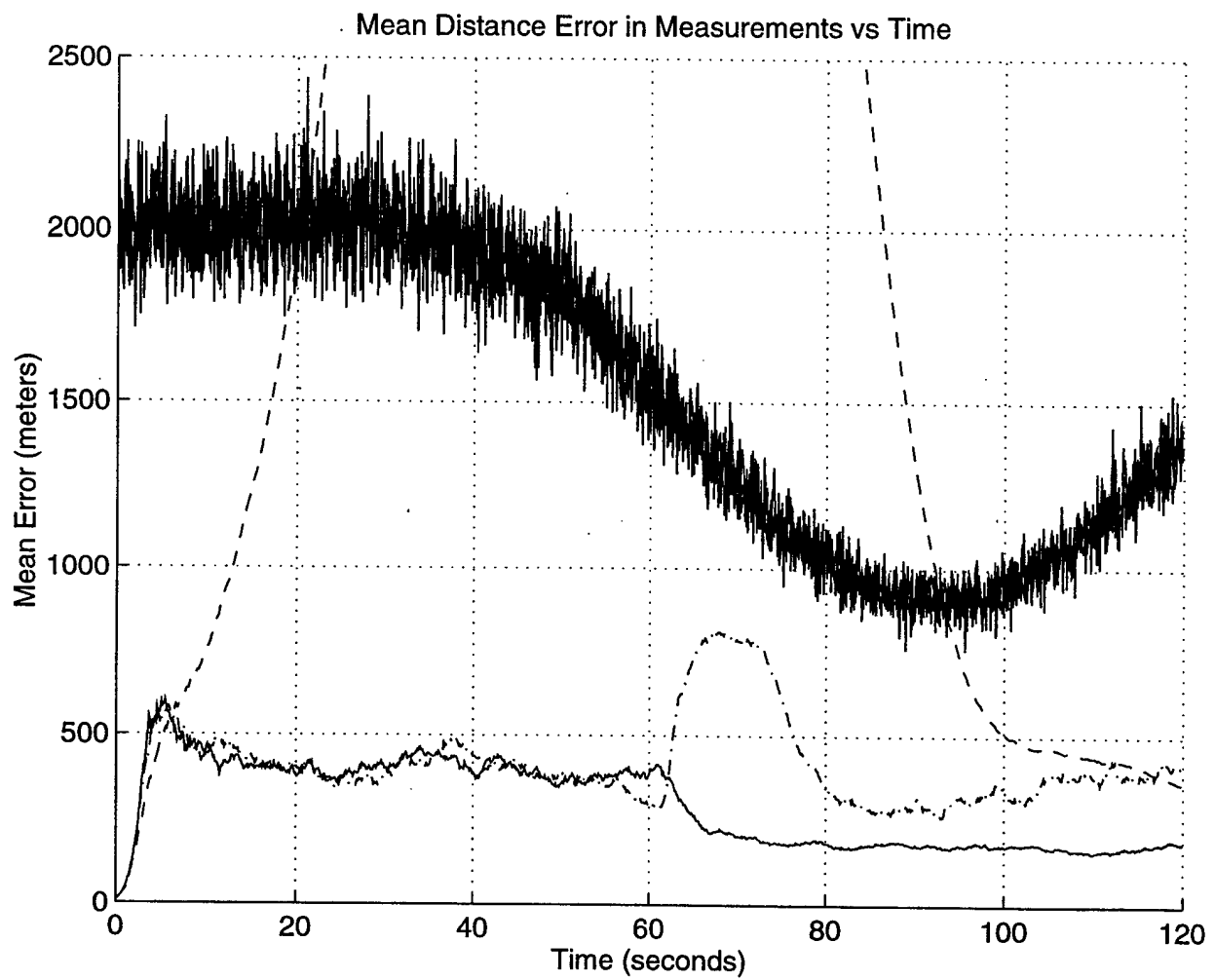


Figure 5.5(b) Comparison (Zoom) of Mean Distance Error (100 Runs).

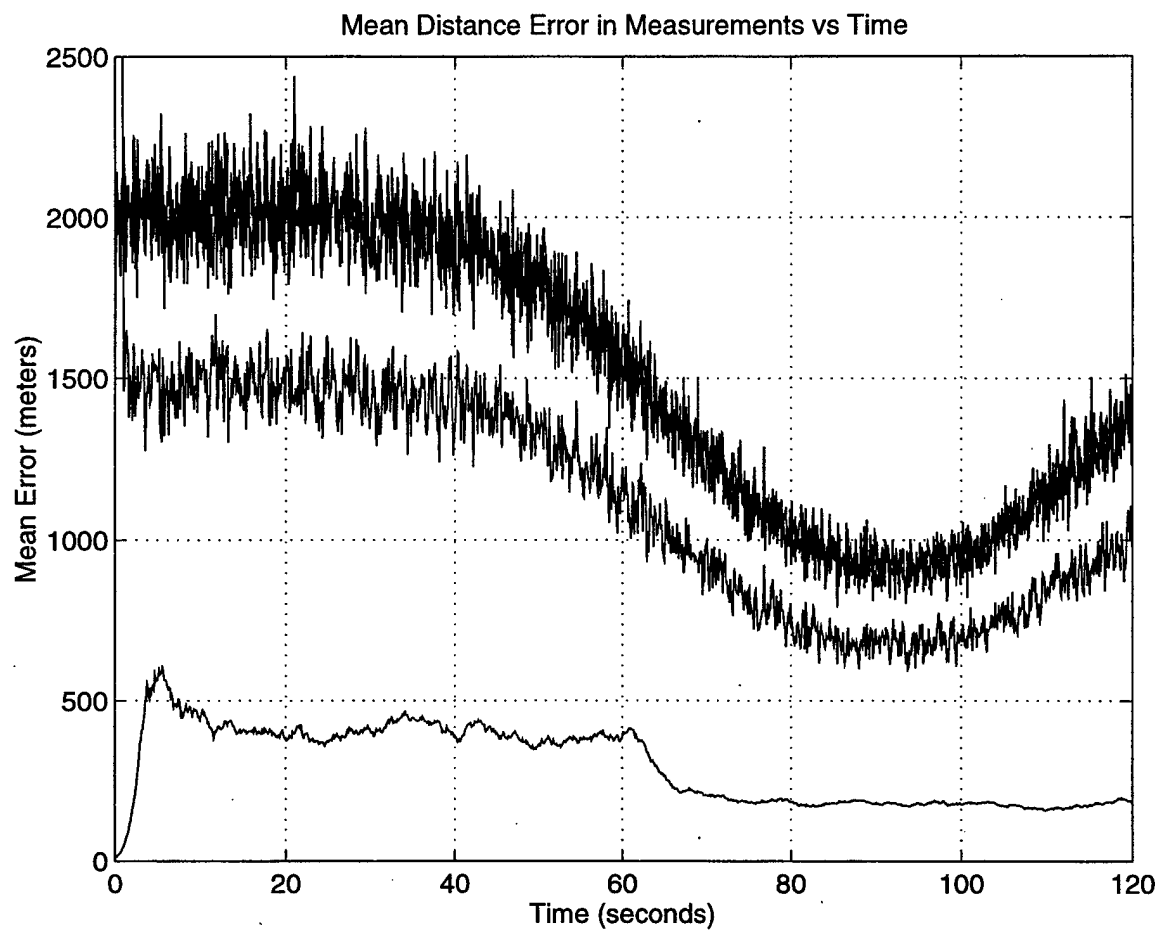


Figure 5.6 Mean Distance Error - IMM vs. $\alpha\beta\gamma$ Tracker (100 Runs).

The final analysis of the IMM algorithm investigates the effect of using a constant switching parameter to switch between the two models of the IMM algorithm. In the previous examples, the process of switching between the two models is controlled by the sigmoid function described in Equation 5.16. This sigmoid switching function changes ρ_{11} from a value of 1.0 to 0.5 as the missile reaches a predetermined altitude at which booster cut-off is likely to occur. The result is a switching process that anticipates a change between models based on prior knowledge of the booster cut-off altitude. In the event that the altitude at which the booster cut-off occurs is not known, a constant value for ρ_{11} can be used. By setting ρ_{11} equal to a constant value of 0.97 (selected only for illustration purposes), the probability that the missile continues to accelerate from one measurement time to the next is 97 percent. The resulting mean distance error is predicted to be larger than the mean distance error of the IMM algorithm that uses a sigmoid switching process. This is due to the slight uncertainty early in the tracking process, in which the tracking algorithm is unsure whether the missile is initially operating in the accelerating or the ballistic model. As shown in the previous sections, the IMM algorithm utilizing a sigmoid switching process is initially certain the missile is accelerating ($\rho_{11} = 1$); hence, it is predicted this algorithm will lead to lower initial values of mean distance error.

The result of the IMM algorithm using a constant switching probability ($\rho_{11} = 0.97$) throughout the tracking process is shown in Figure 5.7. As expected, the mean distance error is initially large, peaking at approximately 800 meters. A comparison of the results of the IMM algorithm using a constant value for ρ_{11} and the results of the IMM

algorithm using a sigmoid switching process is shown in Figure 5.8. The plot of the mean distance error of the IMM algorithm utilizing a constant switching probability is shown as the dashed line. The plot of the mean distance error of the IMM algorithm utilizing a sigmoid switching function is shown as the solid line. As expected, the mean distance error of the IMM algorithm utilizing a constant switching probability is initially larger than the IMM algorithm utilizing a sigmoid switching process. Because the mean distance error of the IMM algorithm utilizing a sigmoid switching process is significantly lower than the IMM algorithm with constant switching probability, it is considered to be the better overall tracking algorithm.

In the next chapter, the EKF, the α - β - γ and the IMM tracking algorithms are implemented on actual TBM profiles. As in the simulated data, measurement noise is added to the TBM profiles. The algorithms are then tested on the real data, and the tracking accuracy of the algorithms is analyzed.

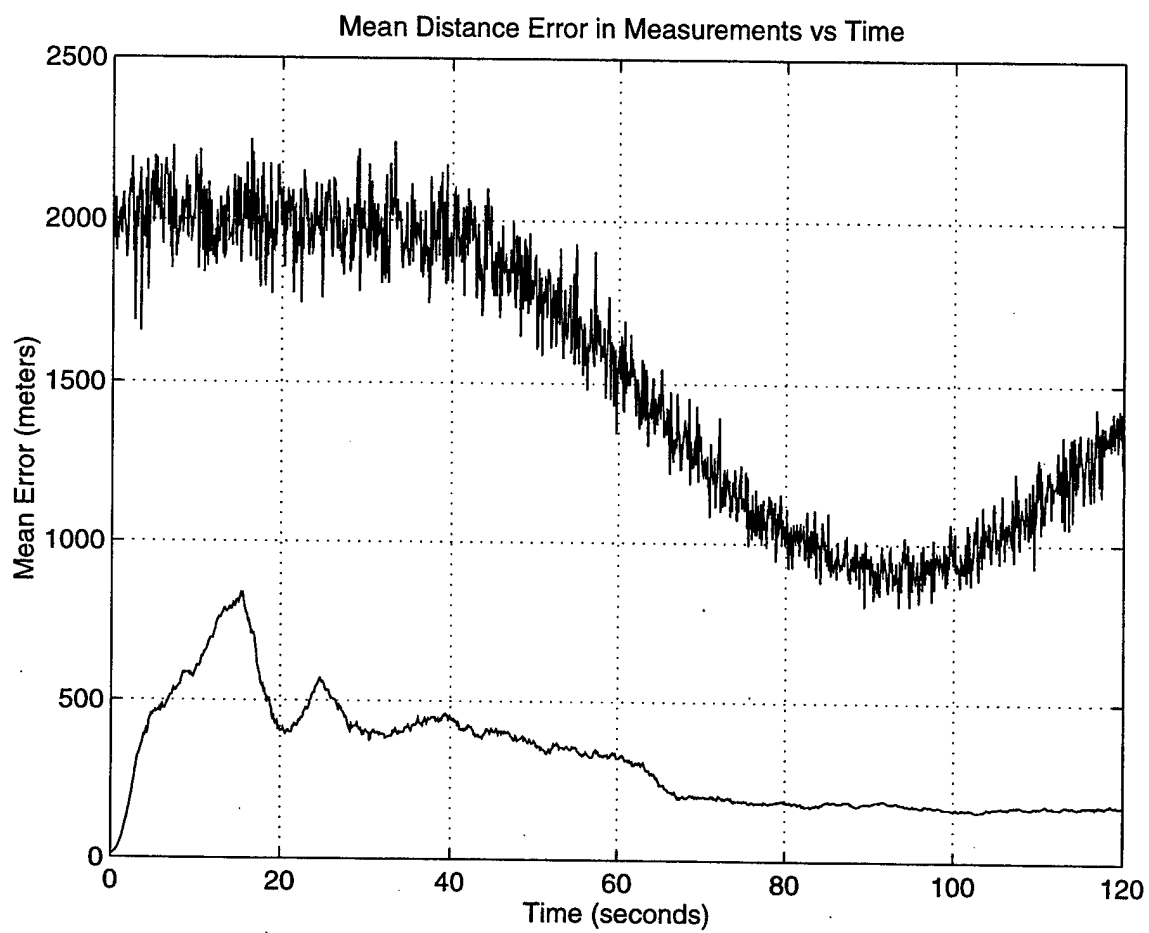


Figure 5.7 Mean Distance Error, IMM with Constant Switching Probability, $\rho_{11}=0.97$, (100 Runs).

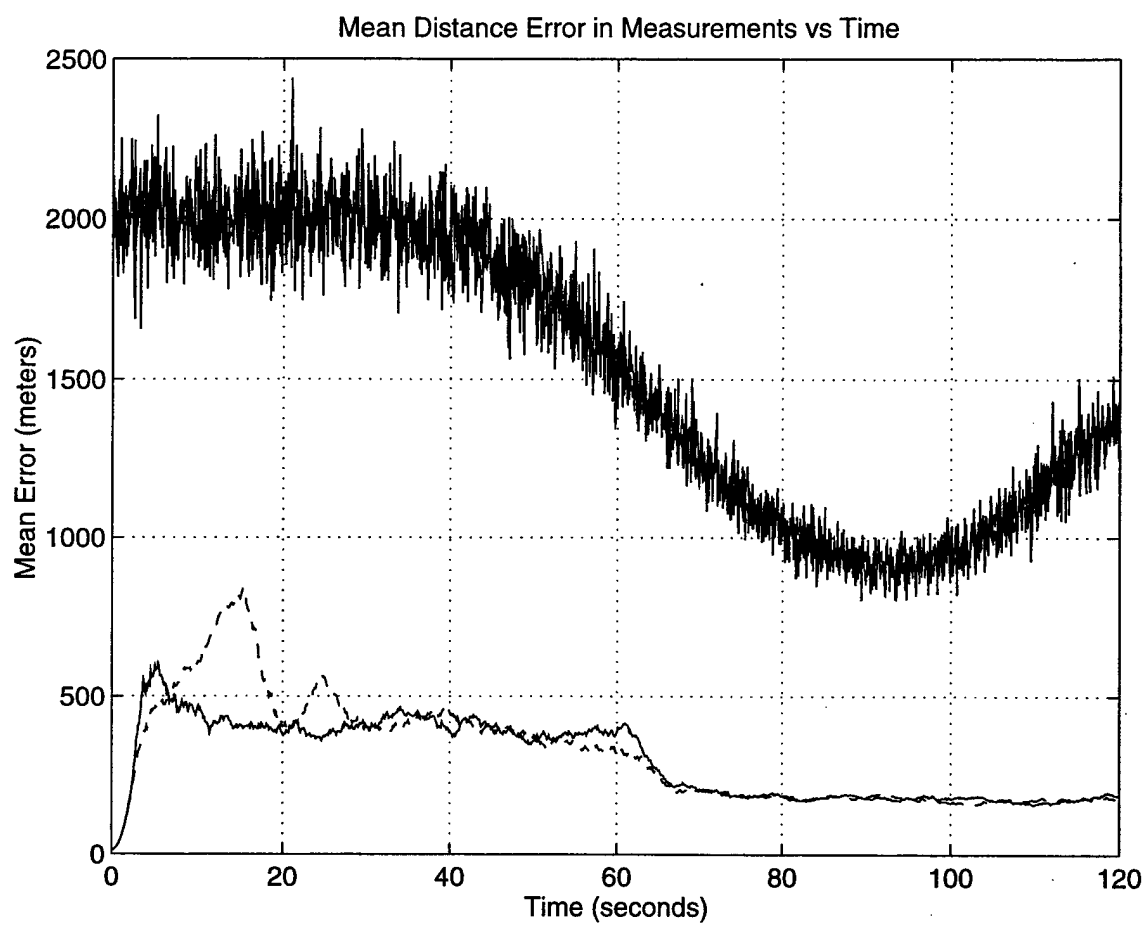


Figure 5.8 IMM with Sigmoid Switching Process vs. IMM with Constant Switching Probability, $\rho_{11}=0.97$ (100 Runs).

VI. ACTUAL TBM PROFILES

In this chapter, the tracking algorithms developed in Chapters III, IV and V, are implemented on actual TBM profiles. The TBM data was graciously given to us by Mr. Thomas Jerardi from the Johns Hopkins Applied Physics Laboratory in Baltimore, Maryland [Ref. 12]. The original source of this TBM data is the National Air Intelligence Center (NAIC) located at Wright-Patterson Air Force Base, Ohio. For security reasons, the specific TBM type is intentionally excluded from the TBM profile data in order to keep this data unclassified. The TBM profile data is provided in Appendix E.

A. TBM PROFILES

A TBM profile is a description of the nominal powered flight trajectory of a given TBM, and an example of a TBM profile is shown in Table 1. A TBM profile consists of an infrared (IR) intensity as a function of time; nominal vertical and horizontal ranges from the launch point as functions of time, and maximum burn time, t_{\max} (62.5 seconds for profile 1 as shown in Table 1)[Ref. 13]. Five TBM profiles are included in Appendix E. Because some of the TBM profiles are very similar, the author has selected TBM profiles 1, 4 and 5 for analysis and discussion in this section. The analysis of TBM profiles 2 and 3 is provided in Appendix E.

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	36.0	0.000	0.000	32	60.6	7.023	3.195
1	36.3	0.006	0.000	33	62.4	7.469	3.491
2	36.6	0.026	0.000	34	64.2	7.928	3.803
3	36.9	0.058	0.000	35	66.0	8.402	4.132
4	37.2	0.103	0.000	36	68.4	8.890	4.479
5	37.5	0.163	0.001	37	70.8	9.393	4.844
6	37.8	0.235	0.004	38	73.2	9.911	5.229
7	38.1	0.322	0.010	39	75.6	10.444	5.633
8	38.4	0.423	0.020	40	78.0	10.992	6.057
9	38.7	0.537	0.036	41	81.2	11.556	6.502
10	39.0	0.666	0.058	42	84.4	12.136	6.969
11	39.5	0.809	0.087	43	87.6	12.732	7.459
12	40.0	0.965	0.124	44	90.8	13.345	7.973
13	40.5	1.136	0.171	45	94.0	13.975	8.511
14	41.0	1.321	0.226	46	96.0	14.622	9.075
15	41.5	1.520	0.292	47	98.0	15.288	9.665
16	42.0	1.733	0.367	48	100.0	15.972	10.282
17	42.5	1.962	0.453	49	102.0	16.675	10.928
18	43.0	2.204	0.550	50	104.0	17.397	11.604
19	43.5	2.460	0.658	51	104.6	18.140	12.309
20	44.0	2.731	0.777	52	105.2	18.904	13.045
21	45.0	3.015	0.908	53	105.8	19.690	13.813
22	46.0	3.312	1.050	54	106.4	20.499	14.613
23	47.0	3.623	1.205	55	107.0	21.332	15.446
24	48.0	3.948	1.372	56	106.4	22.190	16.314
25	49.0	4.286	1.551	57	105.8	23.075	17.217
26	50.6	4.637	1.744	58	105.2	23.986	18.155
27	52.2	5.001	1.950	59	104.6	24.925	19.131
28	53.8	5.378	2.170	60	104.0	25.894	20.145
29	55.4	5.769	2.404	61	98.0	26.894	21.199
30	57.0	6.174	2.652	62	80.0	27.925	22.293
31	58.8	6.591	2.916	62.5	20.0	28.450	22.850

Table 1. Sample TBM Profile (Profile 1) [Ref.12].

B. TBM PROFILE 1

In this section, measurement noise is added to the TBM profile 1 and the α - β - γ , EKF and IMM tracking algorithms are implemented on this trajectory. The mean distance error is calculated for each algorithm and the resulting plots are compared amongst the three filters. To start the analysis, a plot of the actual TBM trajectory is shown in Figure 6.1.

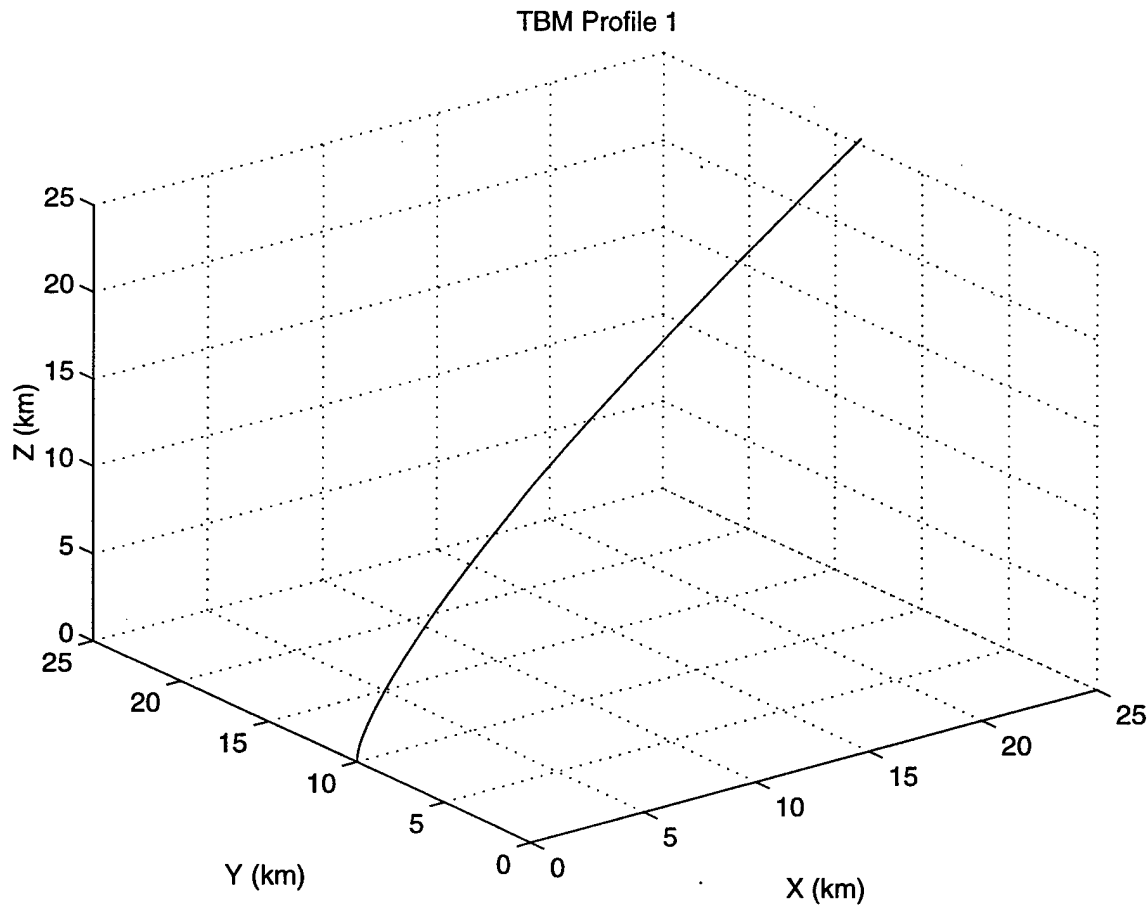


Figure 6.1 TBM Trajectory (Profile 1).

1. Alpha-Beta-Gamma Tracker Results

As in the simulated data, measurement noise is added to the TBM trajectory to simulate a sensor platform observing the missile. In addition, the same sensor position and measurement uncertainties are applied to the TBM profile. Figure 6.2(a) shows a plot of the TBM profile 1 with added measurement noise.

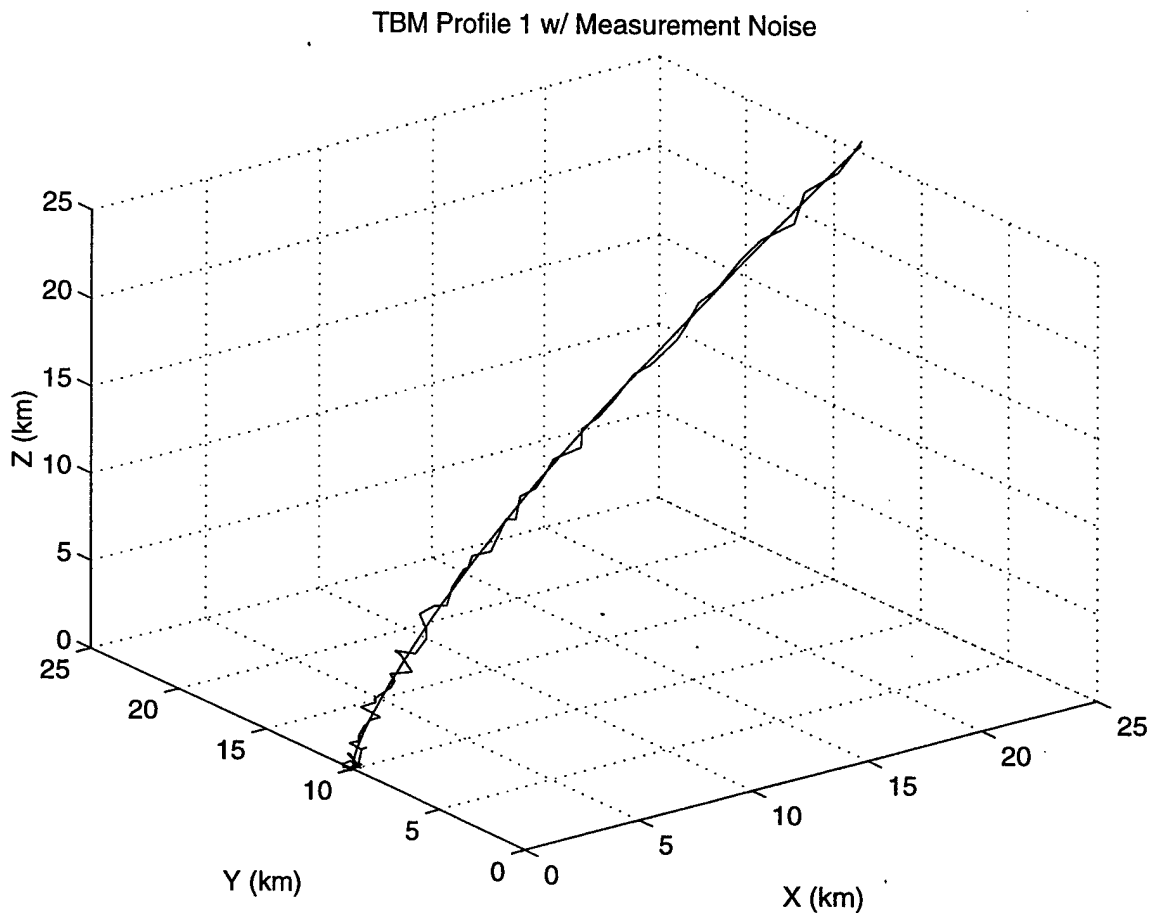


Figure 6.2(a) TBM Trajectory (Profile 1) with Measurement Noise, 100 Runs.

The result of the α - β - γ tracking algorithm is shown in Figure 6.2(b) with the filtered trajectory superimposed on the TBM trajectory for profile 1. These results are obtained over 100 simulation runs, with $\alpha=0.6$.

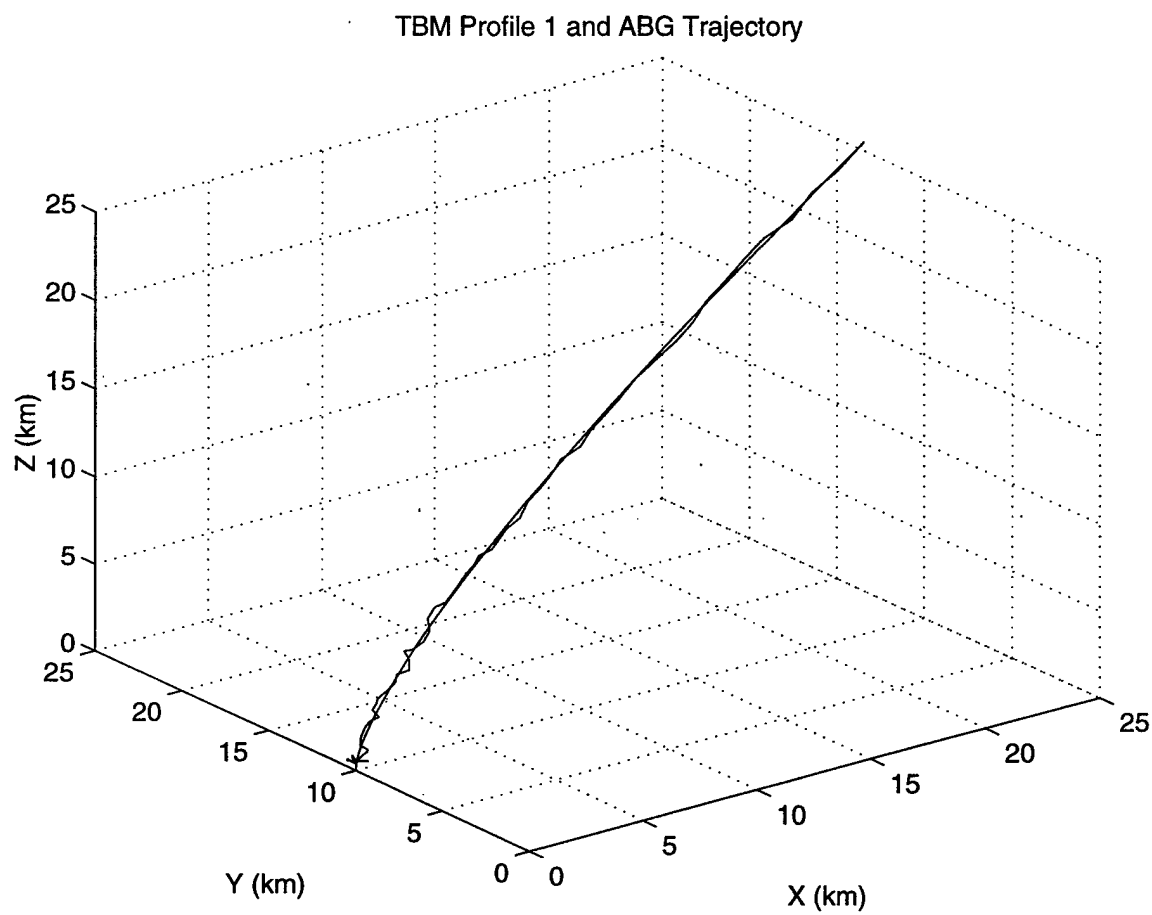


Figure 6.2(b) TBM Trajectory (Profile 1) and α - β - γ Trajectory, $\alpha=0.6$, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.2(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot, shown with a large initial spike, is the mean distance error using the α - β - γ tracking algorithm. These results indicate that the α - β - γ tracker reduces the mean measurement noise by approximately 30 percent, despite a large transient error which is present in the first 10 seconds of the filter. This is shown in Figure 6.2(c) as a spike that peaks to approximately 9,900 meters.

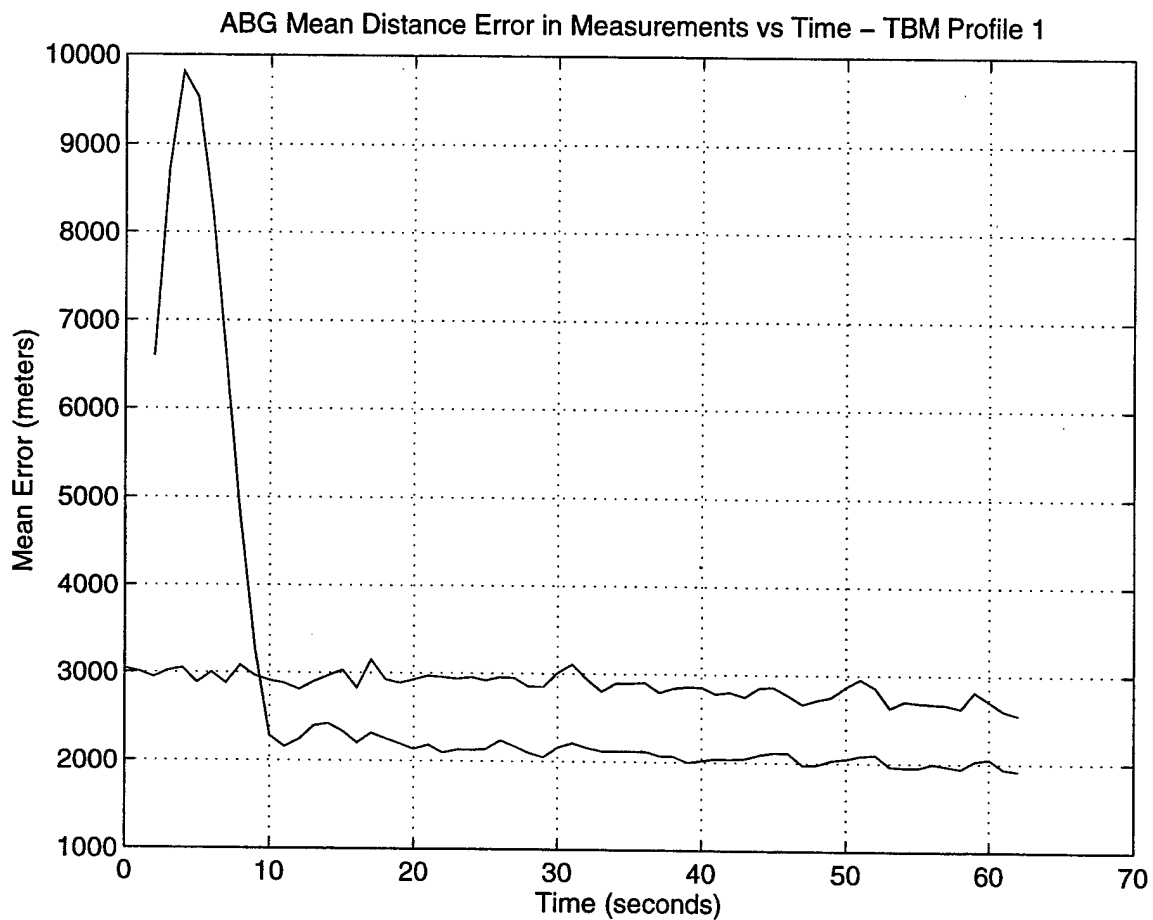


Figure 6.2(c) α - β - γ Tracker (Profile 1) Mean Distance Error, $\alpha=0.6$, 500 Runs.

2. EKF (Accelerating Model) Results

Figure 6.3(a) shows the TBM trajectory for profile 1 with added measurement noise. The result of the EKF (accelerating model) algorithm is shown in Figure 6.3(b) with the filtered trajectory superimposed on the TBM trajectory for profile 1. These results are obtained over 100 simulation runs, with $q^2=10$.

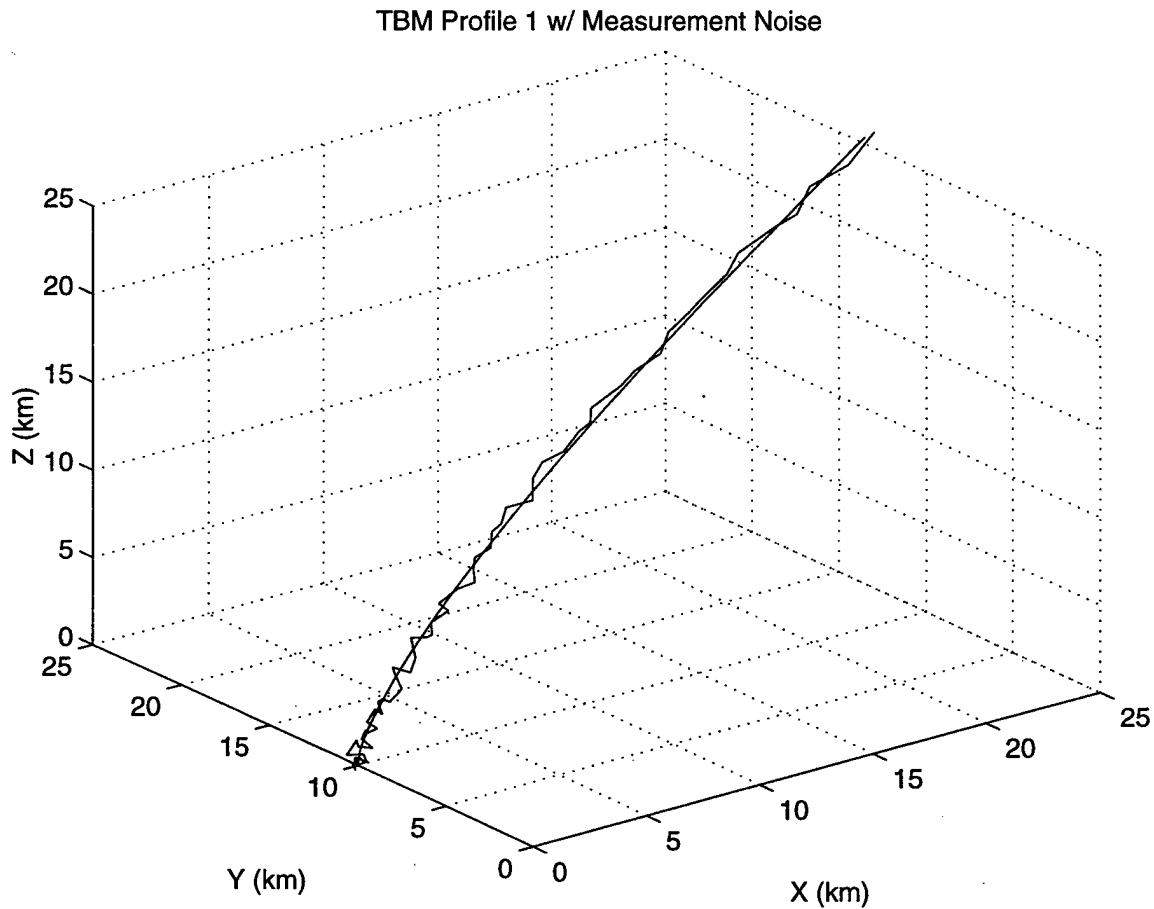


Figure 6.3(a) TBM Trajectory (Profile 1) with Measurement Noise, 100 Runs.

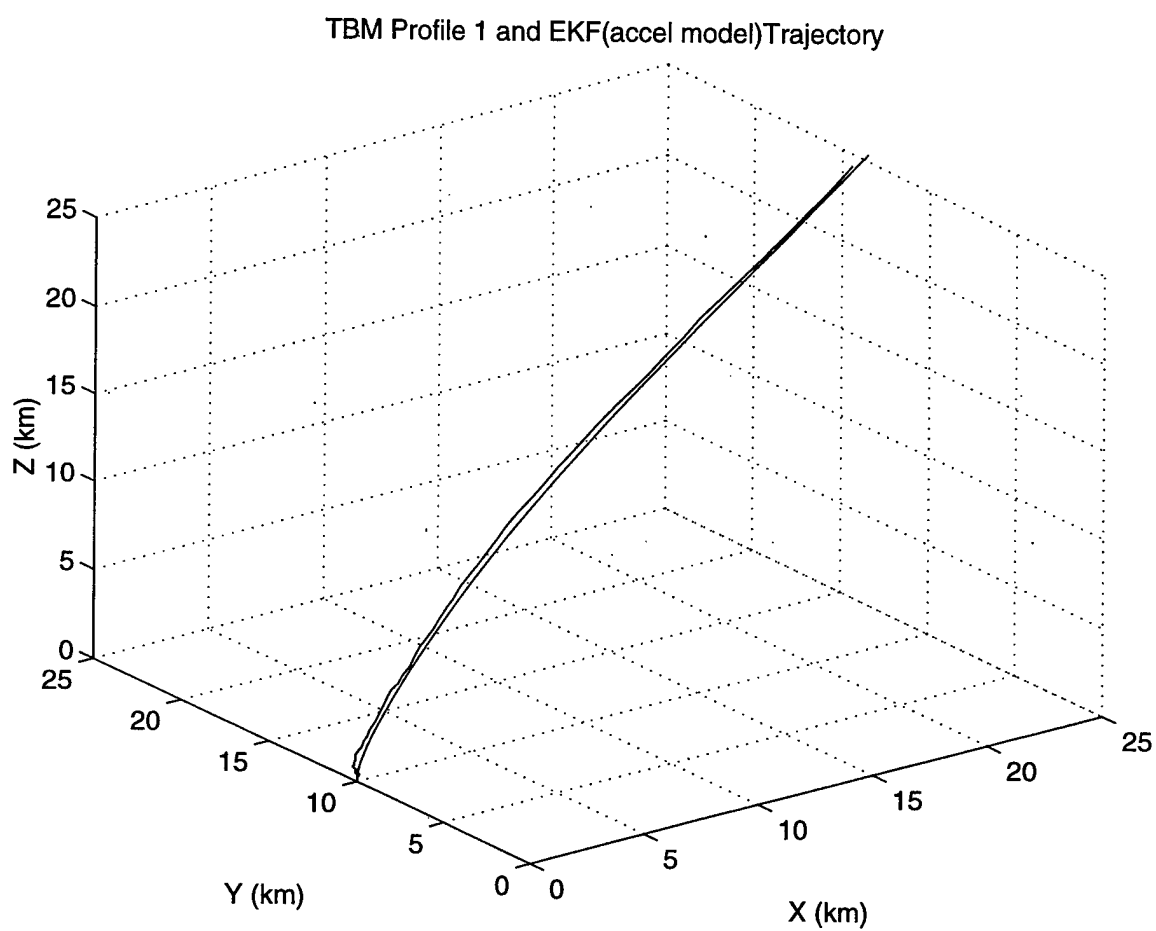


Figure 6.3(b) TBM Trajectory (Profile 1) and EKF Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.3(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the EKF tracking algorithm. These results indicate that the EKF algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1750 meters.

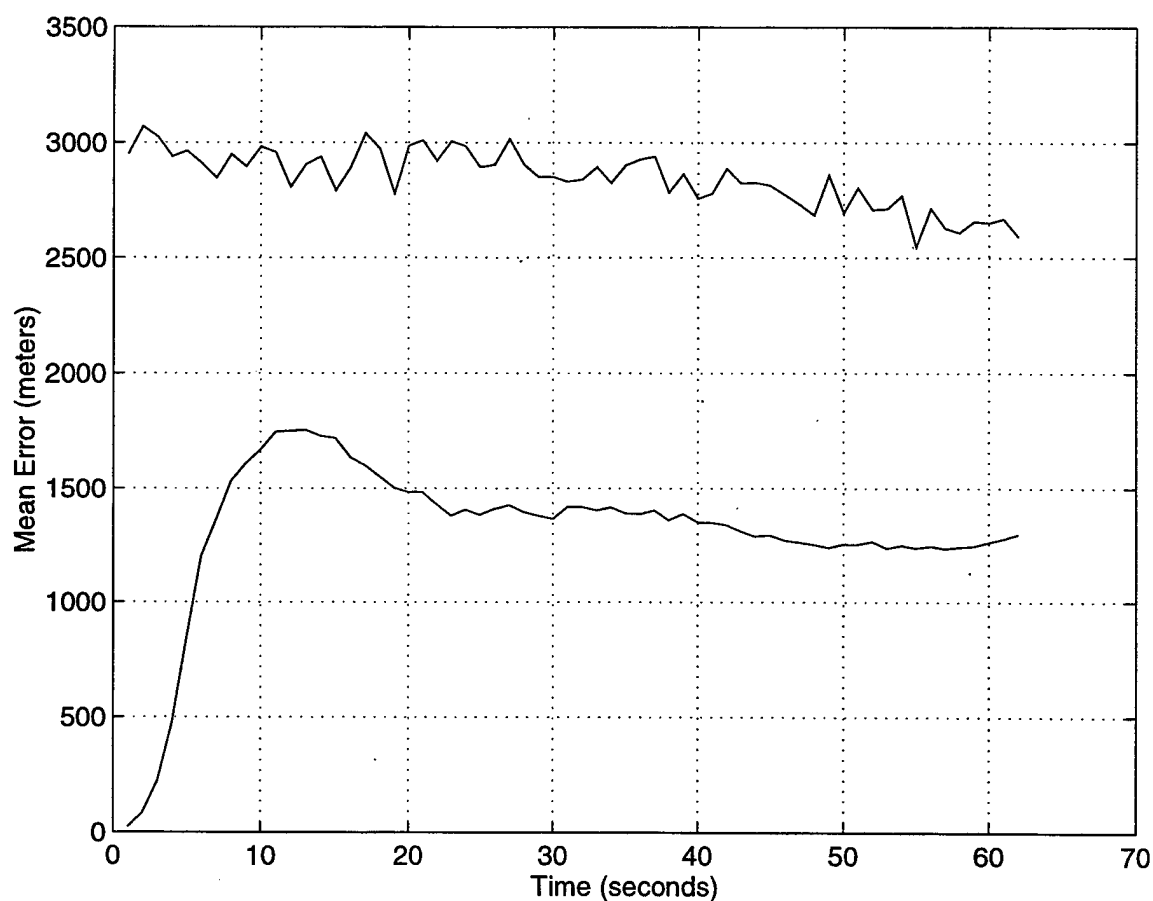


Figure 6.3(c) EKF (Profile 1) Mean Distance Error, 500 Runs.

3. IMM Results

Figure 6.4(a) shows the TBM trajectory for profile 1 with added measurement noise. The result of the IMM algorithm is shown in Figure 6.4(b) with the filtered trajectory superimposed on the TBM trajectory for profile 1. These results are obtained over 100 simulation runs, with $q^2=10$. The switching process is modeled using a sigmoid function (defined in Equation 5.16) that switches element p_{11} , the probability that the missile continues accelerating at the next measurement time, from a value of 1.0 to 0.3. The altitude at the maximum burn time in this profile is approximately 28 km, and in this model, the IMM algorithm is set to start anticipating a change from the accelerating to the ballistic model after the missile reaches an altitude of 20 km.

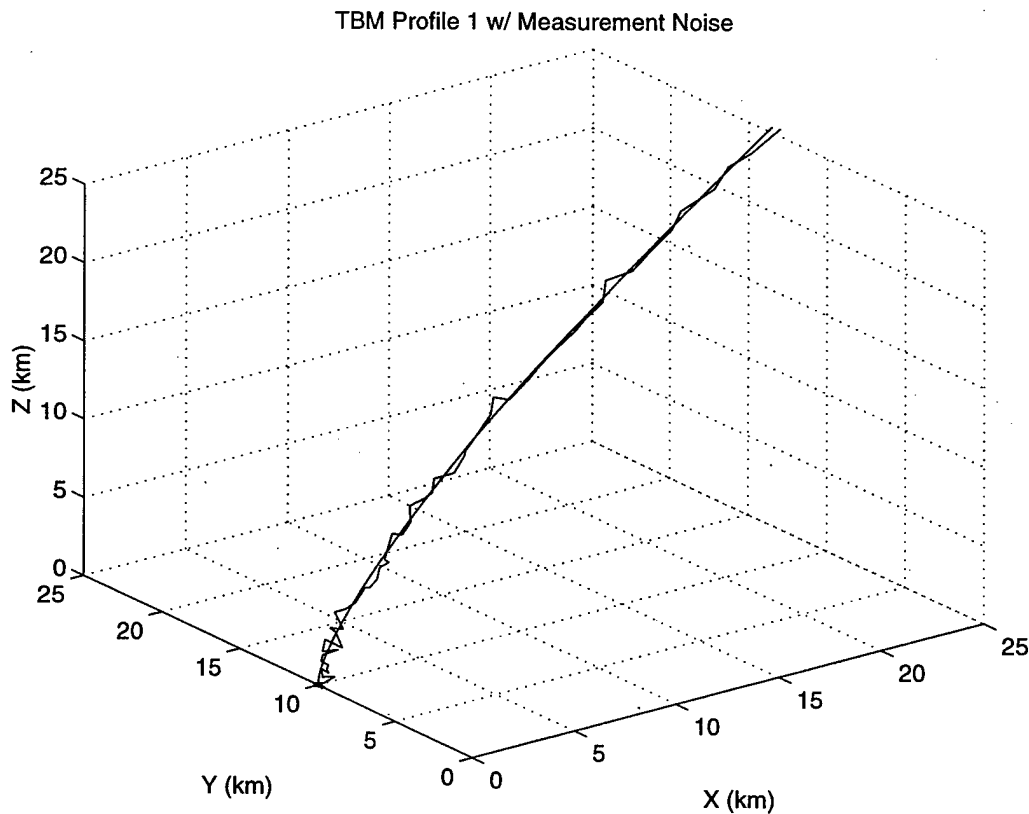


Figure 6.4(a) TBM Trajectory (Profile 1) with Measurement Noise, 100 Runs.

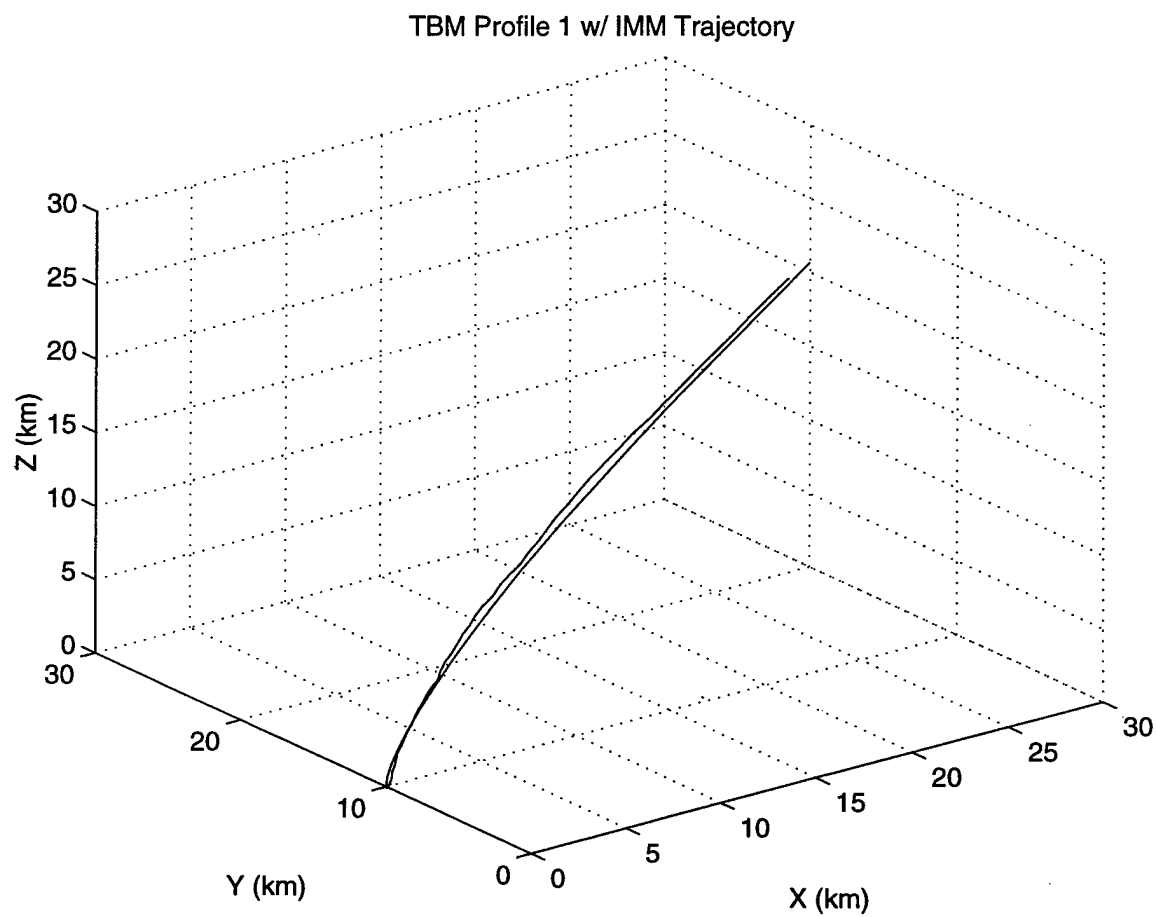


Figure 6.4(b) TBM Trajectory (Profile 1) and IMM Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.4(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the IMM tracking algorithm. These results indicate that the IMM algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1700 meters.

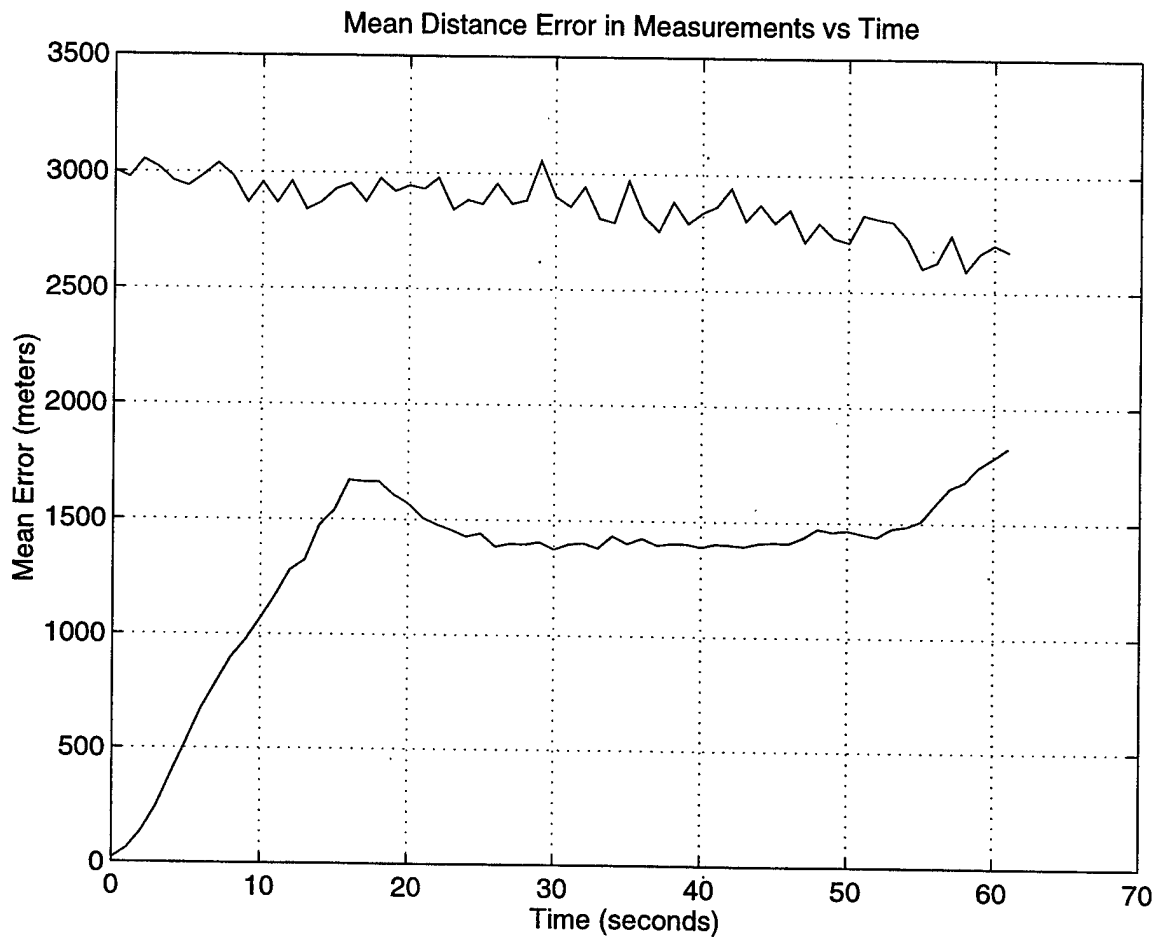


Figure 6.4(c) IMM (Profile 1) Mean Distance Error, 500 Runs.

4. Comparison of Mean Distance Error

Figure 6.5(a) shows a comparison of the mean distance error plots for the α - β - γ tracker (shown as a dash-dot line), the EKF accelerating model (shown as a dashed line), and the IMM algorithm (shown as a solid line). Figure 6.5(b) shows a close-up of the comparison. Since the TBM profile contains missile positions only up to the maximum burn time, the TBM profile does not contain missile data during the ballistic phase. Therefore, in the IMM algorithm a switch to the ballistic model does not occur, and thus, the EKF and IMM algorithms have similar results except for a small deviation starting at

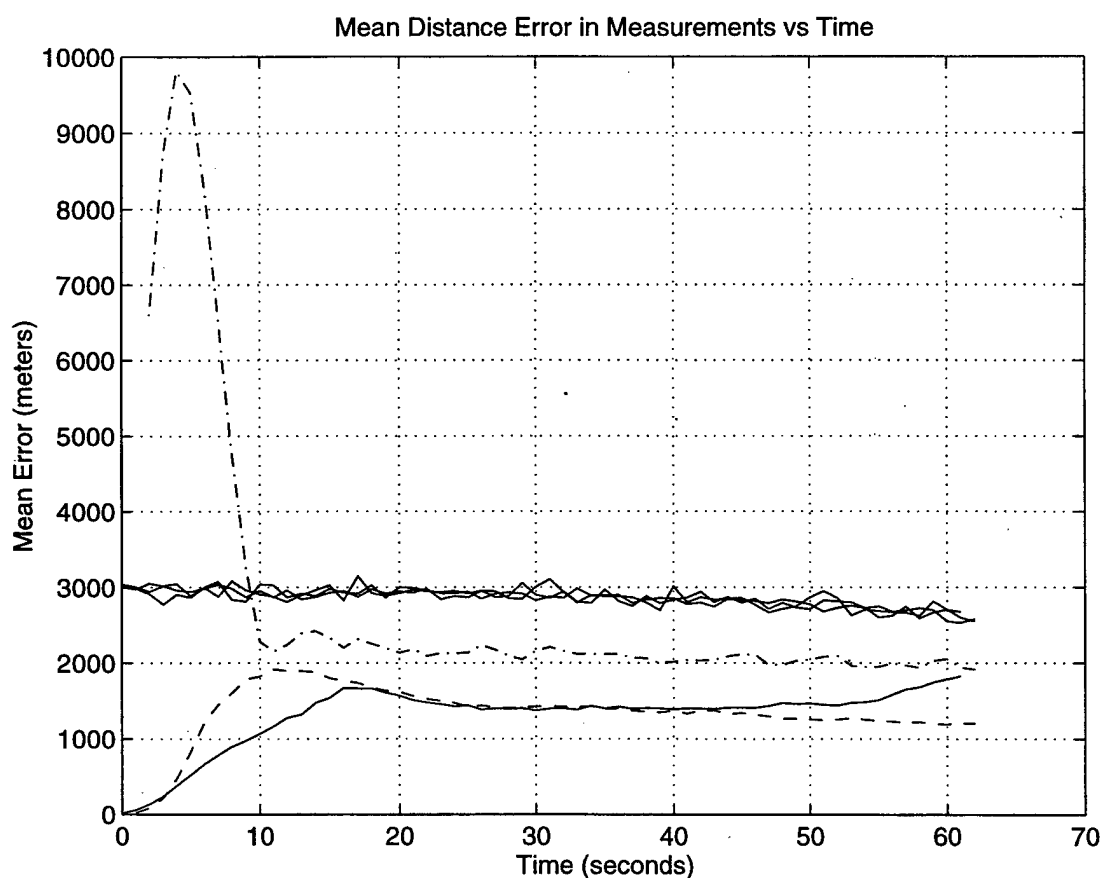


Figure 6.5(a) Comparison of α - β - γ , EKF and IMM Mean Distance Error, 500 Runs.

approximately 50 seconds. This is due to the IMM algorithm anticipating the change from the accelerating model to the ballistic model. The following close-up graph clearly indicates the IMM algorithm anticipates the impending switch to the ballistic phase.

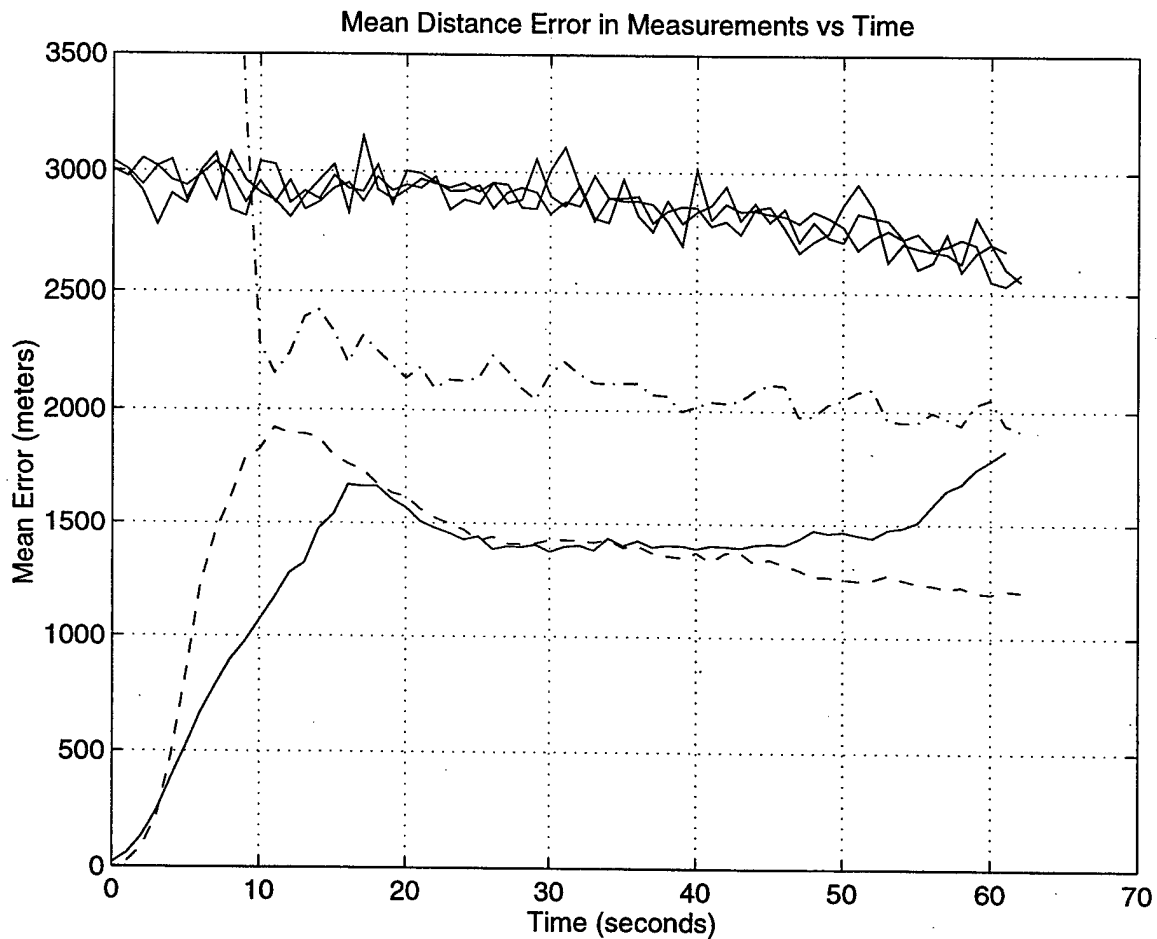


Figure 6.5(b) Comparison (Close-up) of Mean Distance Error, 500 Runs.

As discussed in Chapter V, the IMM algorithm switches between system models by using either a sigmoid switching function or by using a constant value for the switching probability, ρ_{11} . The previous example used a sigmoid switching function that changed the value of ρ_{11} as the missile reached a predetermined altitude when booster cut-off was likely to occur. In the event that this altitude is not known, a constant value for ρ_{11} can be used. For comparison purposes, the IMM algorithm is implemented on the actual TBM data using a constant switching probability, with $\rho_{11}=0.75$. Figure 6.5(c) shows a comparison of the mean distance error for the IMM algorithm utilizing a sigmoid switching function (shown as a solid line) and the IMM algorithm utilizing a constant switching probability (shown as a dashed line). As in the simulated data in Chapter V, the mean distance error of the IMM algorithm utilizing a constant switching probability is slightly larger early in the tracking process (although not as pronounced as in the simulated data). This is due to the slight uncertainty in the tracking algorithm, in which the tracking algorithm is unsure whether the missile is initially operating in the accelerating or the ballistic model. Although the IMM algorithm utilizing a sigmoid switching function performs better in the early part of the tracking process, the IMM algorithm utilizing a constant switching probability performs better in the latter part of the tracking process. This graph illustrates a trade-off in performance between the two switching processes.

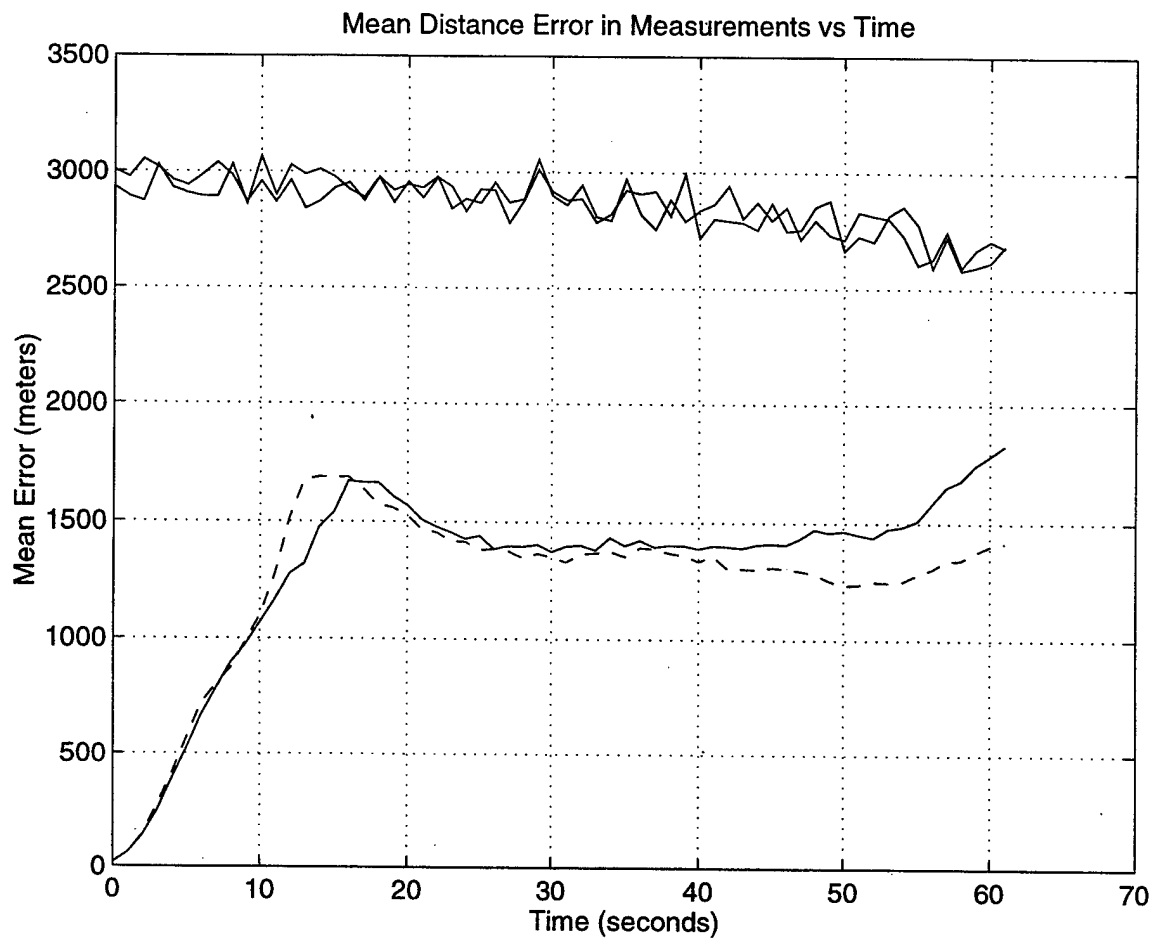


Figure 6.5(c) IMM with Sigmoid Switching Process vs. IMM with Constant Switching Probability, $\rho_{11}=0.75$ (500 Runs).

C. TBM PROFILE 4

As in the previous section, measurement noise is added and the α - β - γ , EKF and IMM tracking algorithms are implemented on actual TBM data. In addition, the mean distance error is computed for each algorithm, and the resulting plots are compared

amongst the three filters for a new set of data called TBM profile 4. The plot of the TBM trajectory for profile 4 is shown in Figure 6.6.

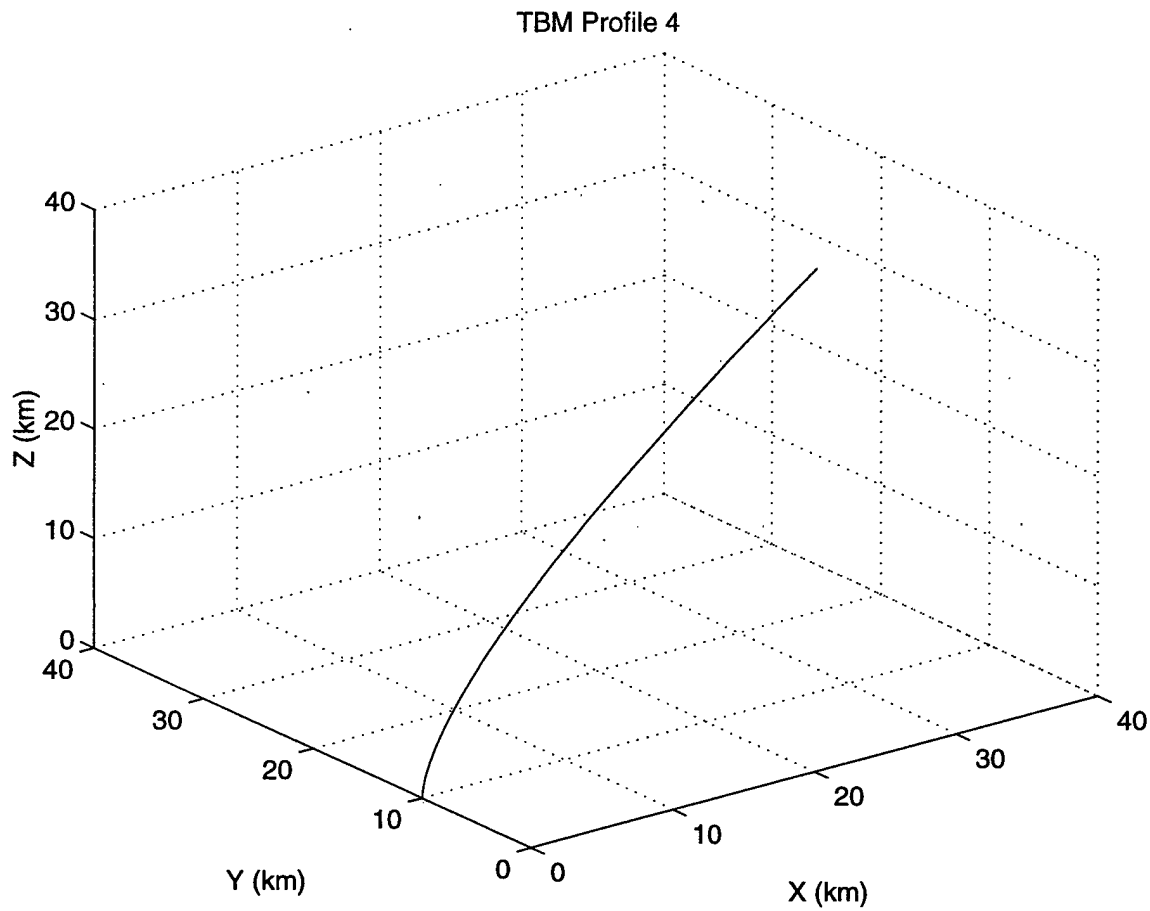


Figure 6.6 TBM Trajectory (Profile 4).

1. Alpha-Beta-Gamma Tracker Results

Figure 6.7(a) shows a plot of the TBM profile 4 with added measurement noise.

The result of the α - β - γ tracking algorithm is shown in Figure 6.7(b) with the filtered trajectory superimposed on the TBM trajectory for profile 4. These results are obtained over 100 simulation runs, with $\alpha=0.6$.

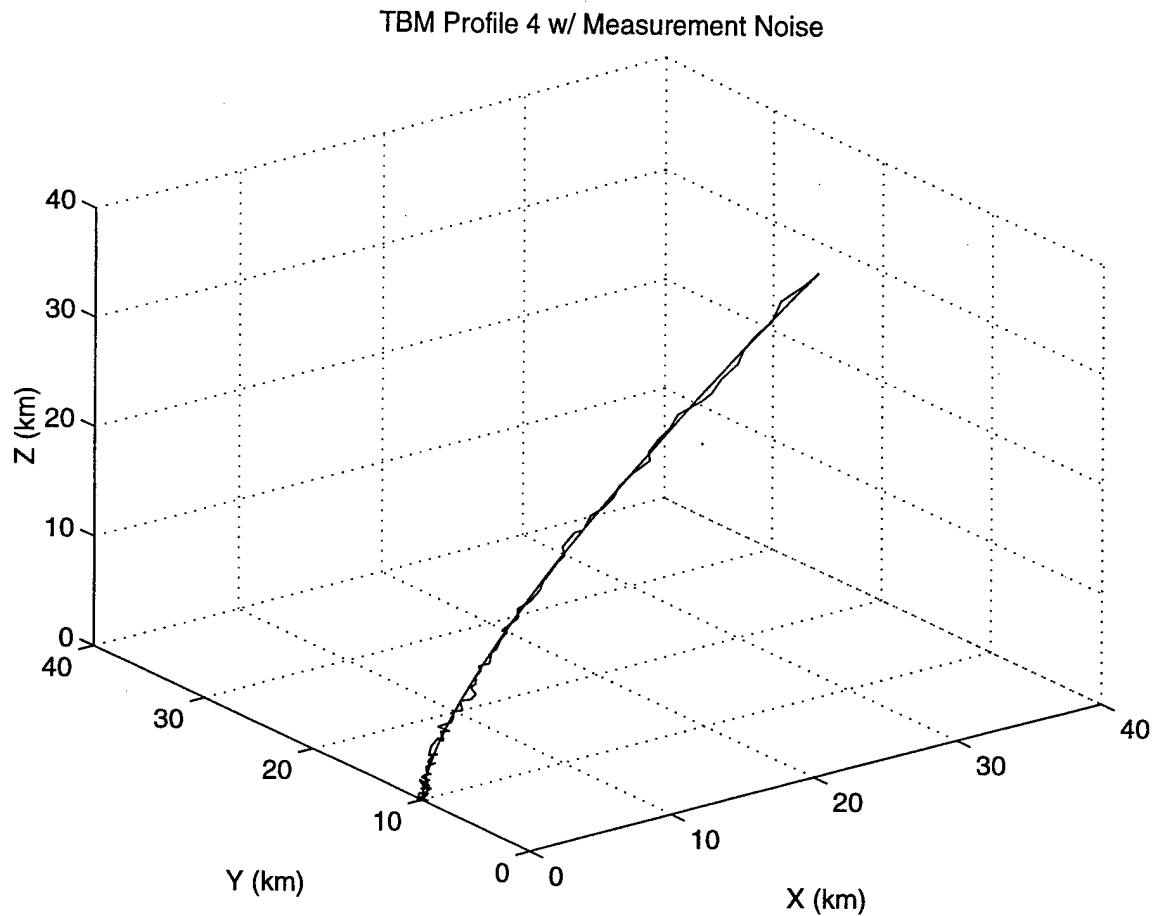


Figure 6.7(a) TBM Trajectory (Profile 4) with Measurement Noise, 100 Runs.

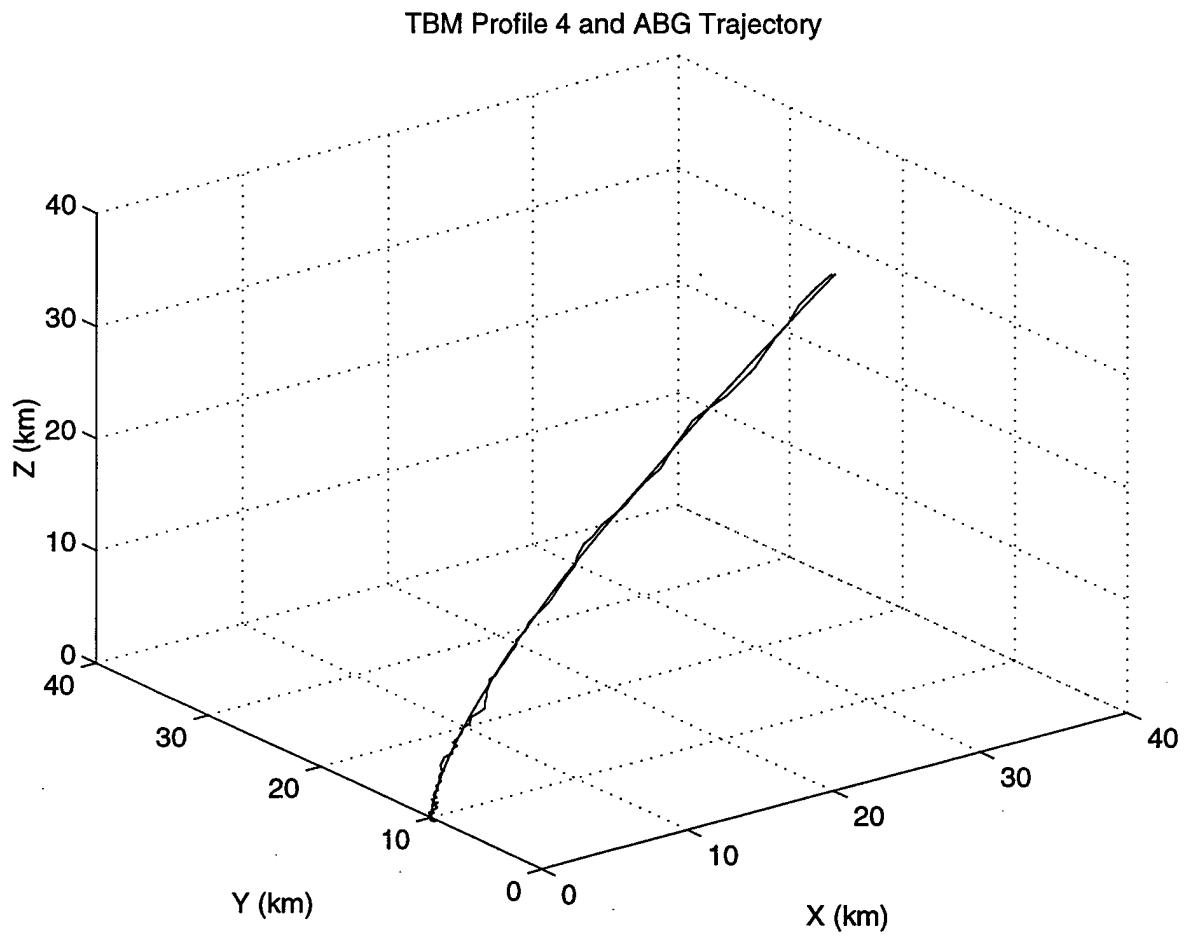


Figure 6.7(b) TBM Trajectory (Profile 4) and α - β - γ Trajectory, $\alpha=0.6$, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.7(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot, shown with a large initial spike, is the mean distance error using the α - β - γ tracking algorithm. These results indicate that the α - β - γ tracker reduces the mean measurement noise by approximately 30 percent despite a large transient error which is present in the first 10 seconds of the filter. The transient error is shown in Figure 6.7(c) as a spike that peaks to approximately 9,700 meters.

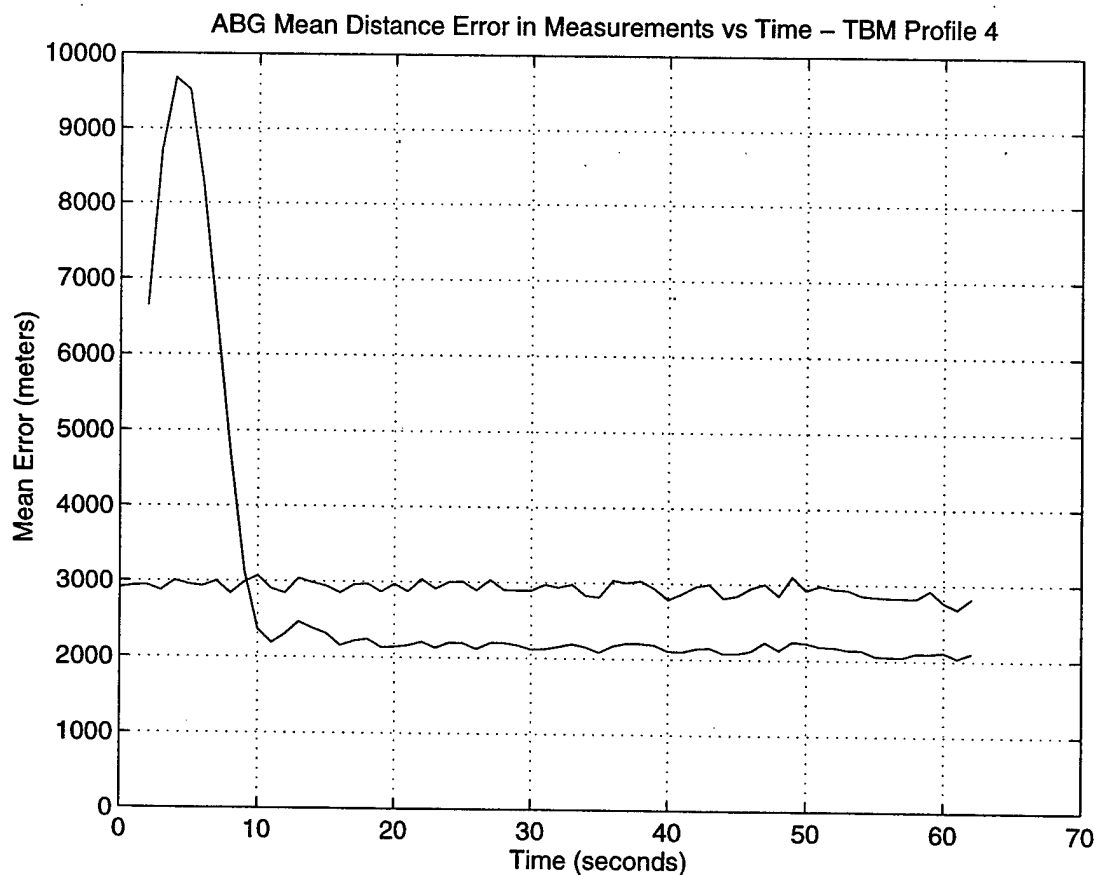


Figure 6.7(c) α - β - γ Tracker (Profile 4) Mean Distance Error, $\alpha=0.6$, 500 Runs.

2. EKF (Accelerating Model) Results

Figure 6.8(a) shows the TBM trajectory for profile 4 with added measurement noise. The result of the EKF (accelerating model) algorithm is shown in Figure 6.8(b) with the filtered trajectory superimposed on the TBM trajectory for profile 4. These results are obtained over 100 simulation runs, with $q^2=10$.

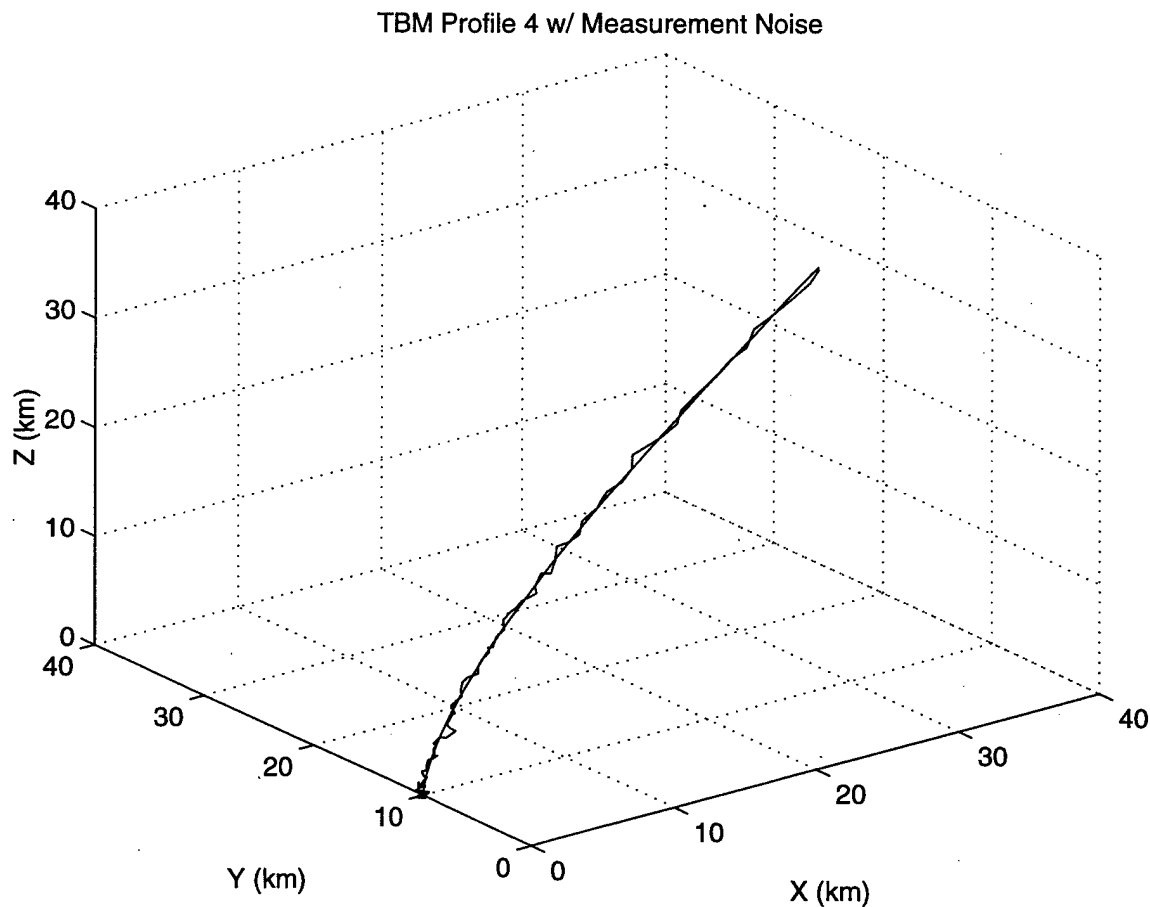


Figure 6.8(a) TBM Trajectory (Profile 4) with Measurement Noise, 100 Runs.

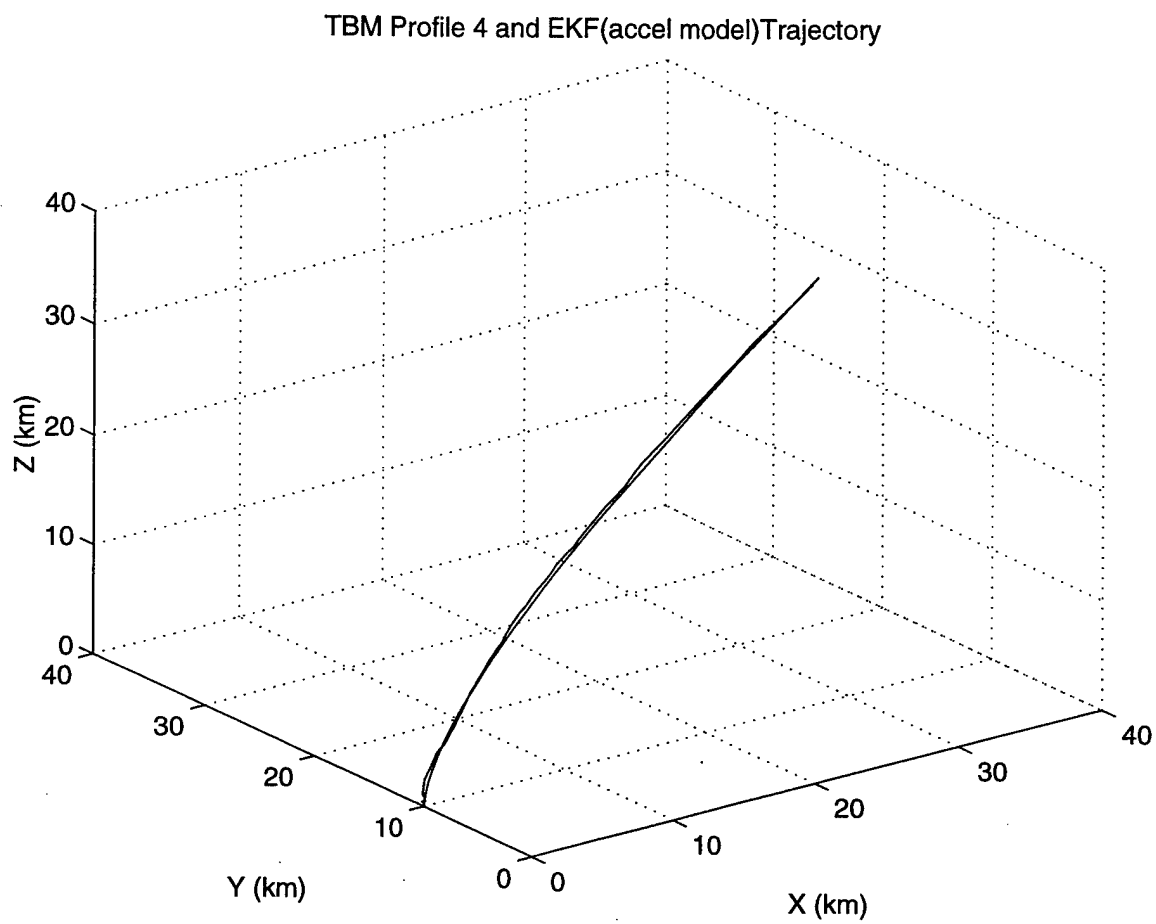


Figure 6.8(b) TBM Trajectory (Profile 4) and EKF Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.8(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the EKF tracking algorithm. These results indicate that the EKF algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1900 meters.

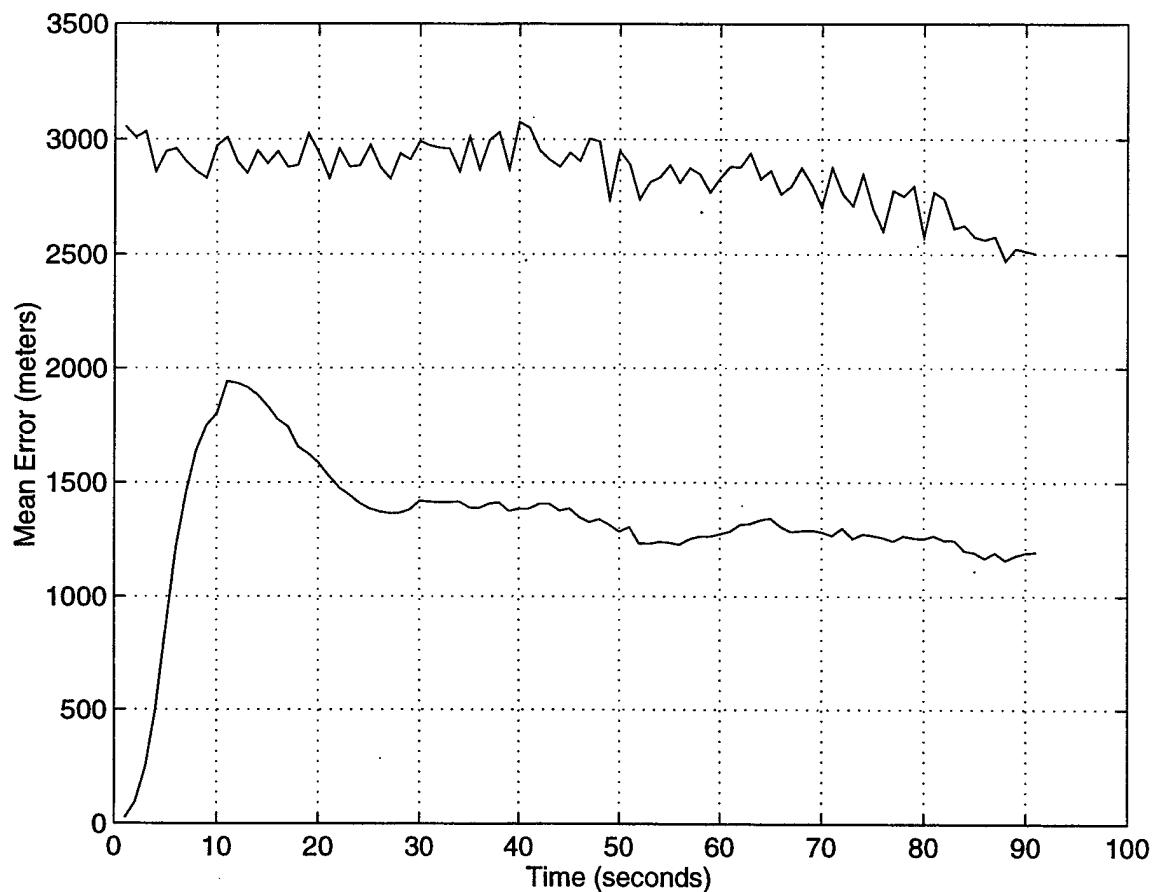


Figure 6.8(c) EKF (Profile 4) Mean Distance Error, 500 Runs.

3. IMM Results

Figure 6.9(a) shows the TBM trajectory for profile 4 with added measurement noise. The result of the IMM algorithm is shown in Figure 6.9(b) with the filtered trajectory superimposed on the TBM trajectory for profile 4. These results are obtained over 100 simulation runs, with $q^2=10$. The switching process is modeled using a sigmoid function that switches element ρ_{11} from a value of 1.0 to 0.5. The altitude at the maximum burn time in this profile is approximately 38 km, and in this model the IMM algorithm is set to start anticipating a change from the accelerating to the ballistic model after the missile reaches an altitude of 32 km.

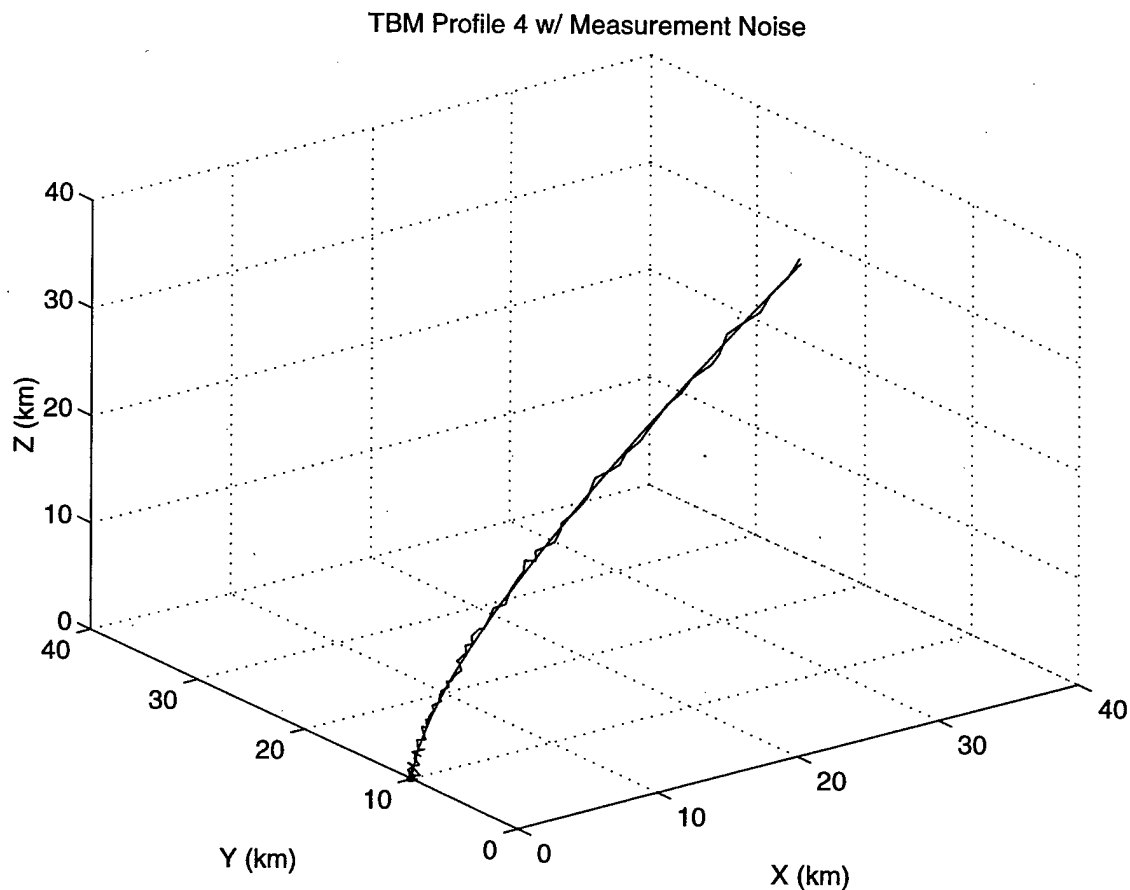


Figure 6.9(a) TBM Trajectory (Profile 4) with Measurement Noise, 100 Runs.

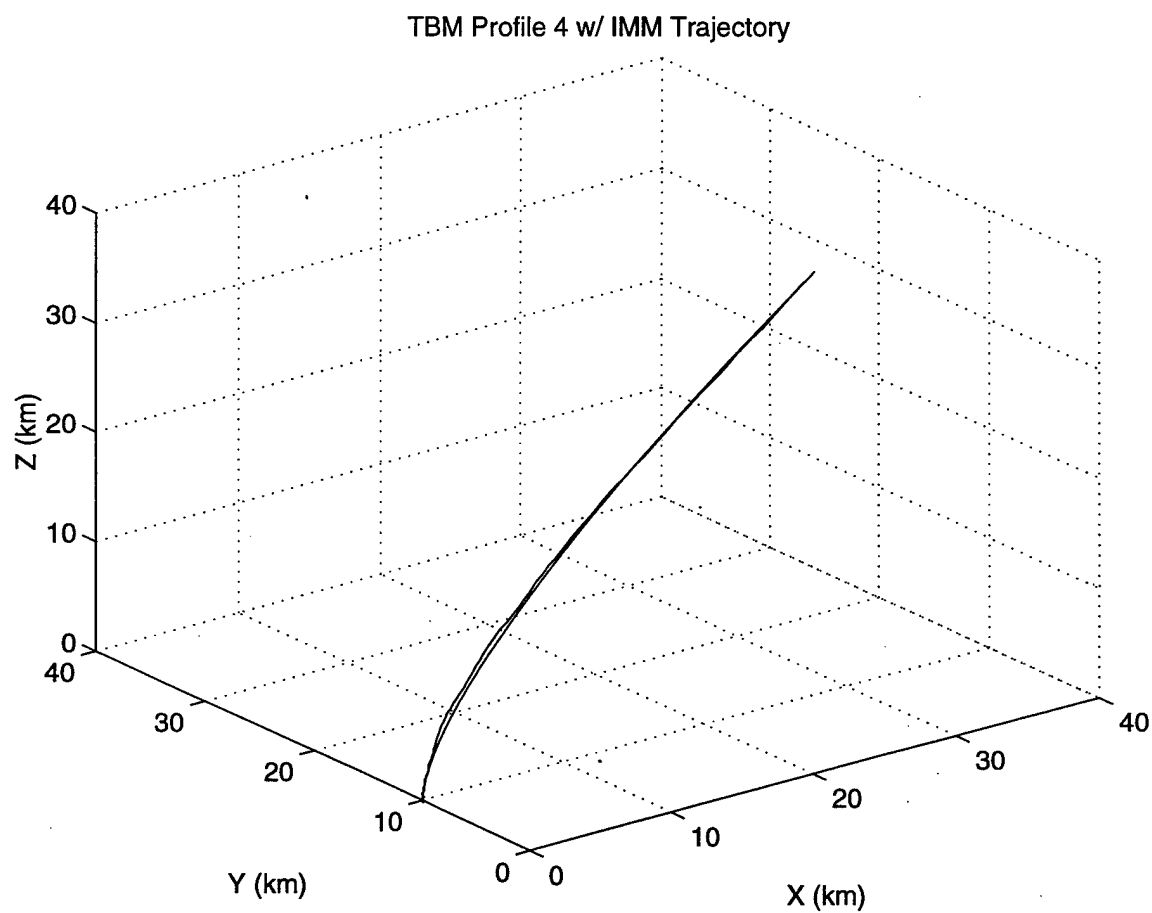


Figure 6.9(b) TBM Trajectory (Profile 4) and IMM Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.9(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the IMM tracking algorithm. These results indicate that the IMM algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1900 meters.

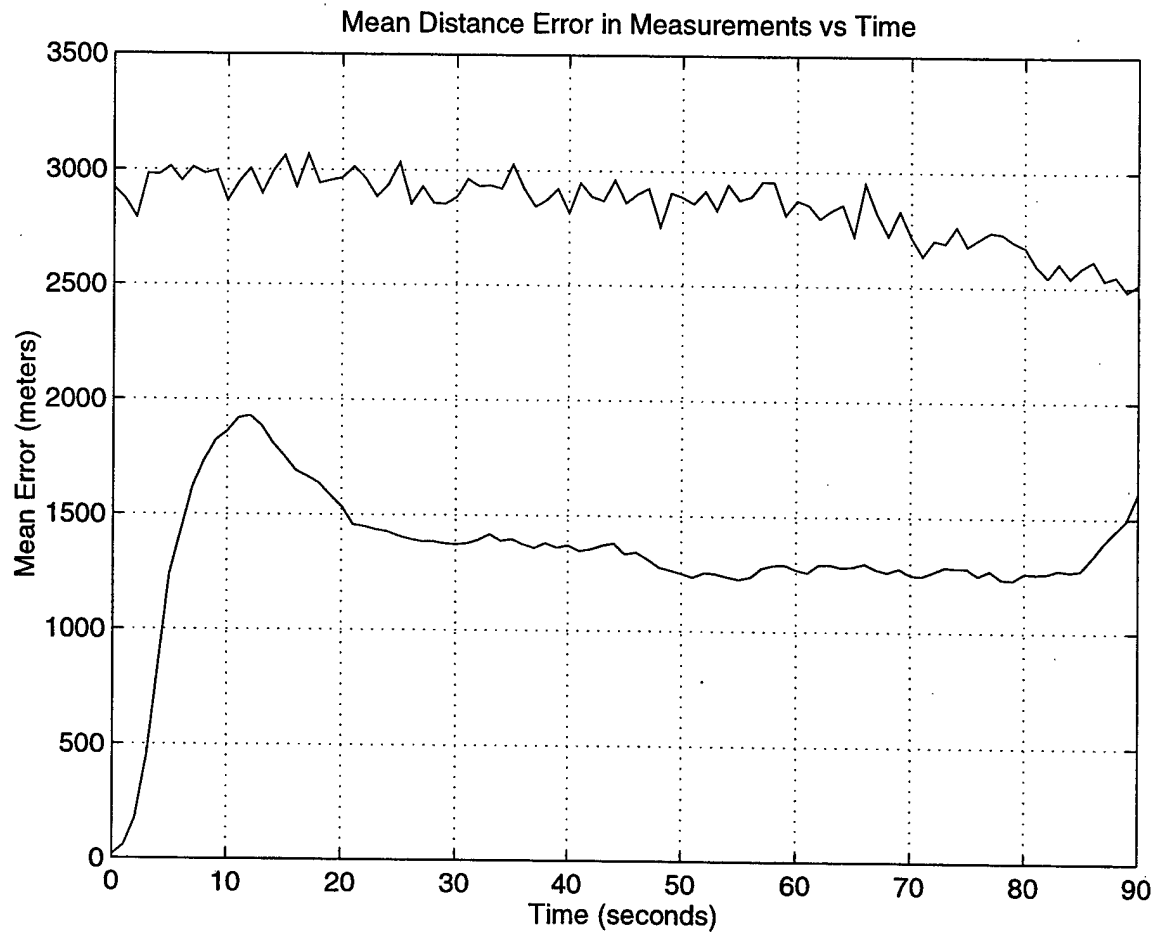


Figure 6.9(c) IMM (Profile 4) Mean Distance Error, 500 Runs.

4. Comparison of Mean Distance Error

Figure 6.10(a) shows a comparison of the mean distance error plots for the α - β - γ tracker (shown as a dash-dot line), the EKF accelerating model (shown as a dashed line), and the IMM algorithm (shown as a solid line). Figure 6.10(b) shows a close-up of the comparison. As expected, the EKF algorithm and the IMM algorithm continue to show similar results since the IMM algorithm does not switch to the ballistic model.

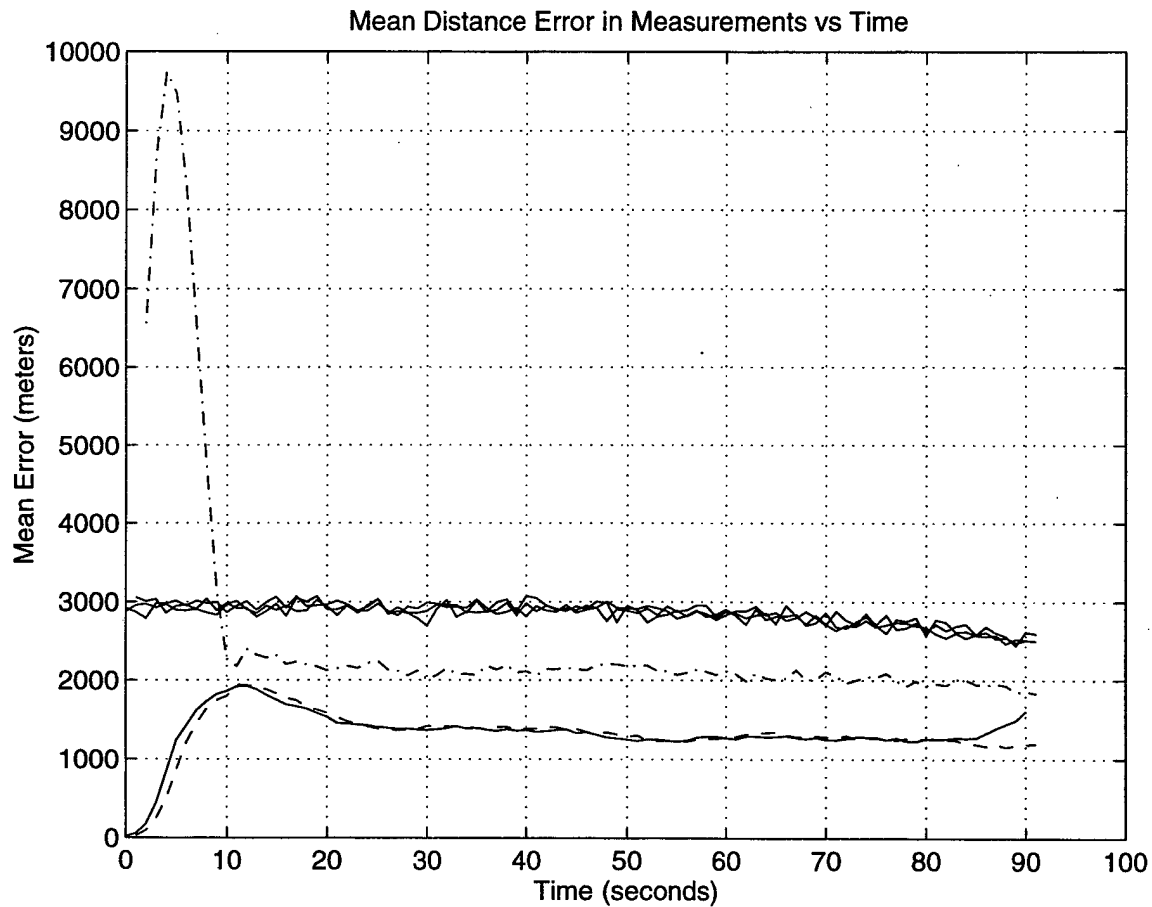


Figure 6.10(a) Comparison of α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

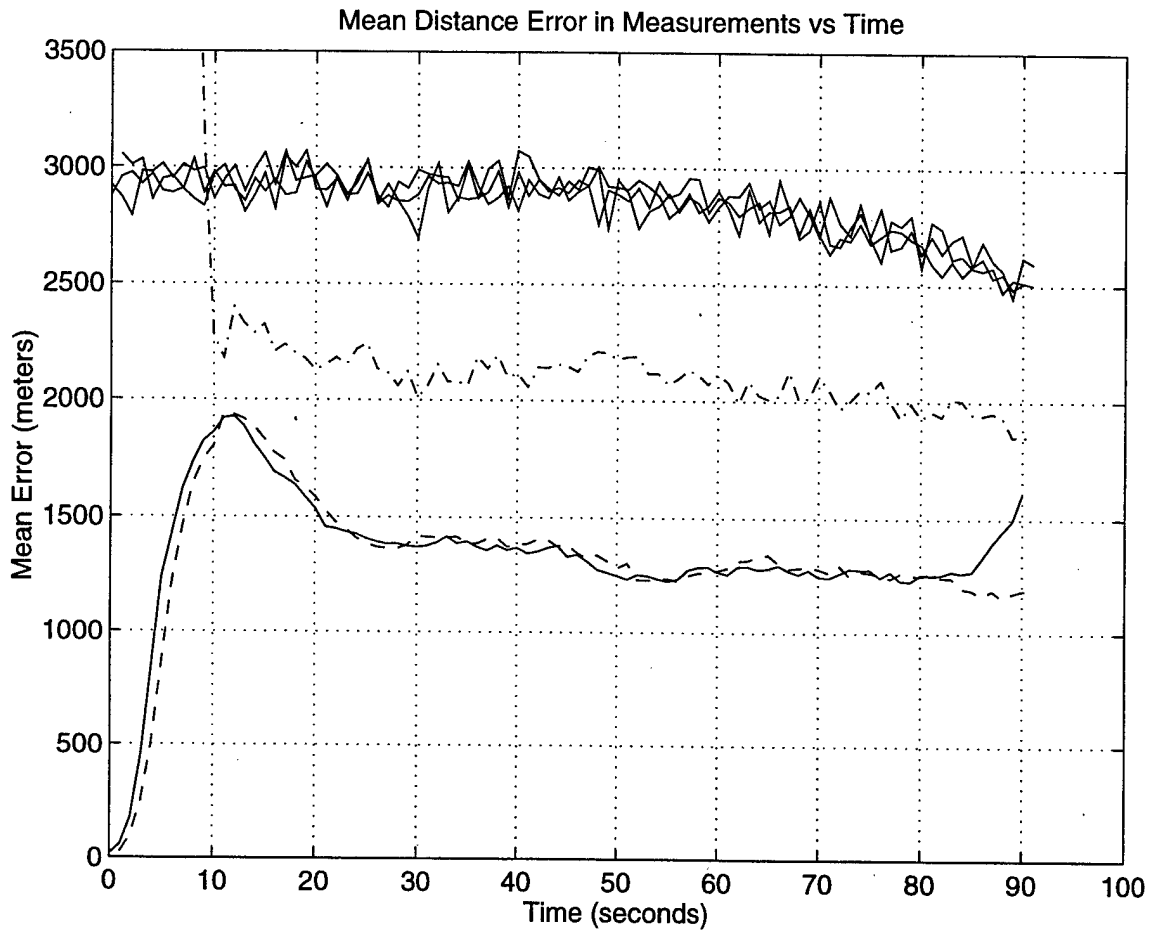


Figure 6.10(b) Comparison (Close-up) of Mean Distance Error, 500 Runs.

As in the previous section, an analysis of the IMM switching processes utilizing a constant switching probability is conducted with $\rho_{11}=0.75$. The results are compared to the previous example, where the IMM algorithm used a sigmoid switching function to change the value of ρ_{11} . Figure 6.10(c) shows a comparison of the mean distance error for the IMM algorithm utilizing a sigmoid switching function (shown as a solid line) and the IMM algorithm utilizing a constant switching probability (shown as a dashed line).

Contrary to the simulated data in Chapter V and the actual data in TBM profile 1, the results of the IMM algorithm utilizing these two switching processes do not behave as expected. In this example, the mean distance error for the IMM algorithm utilizing a constant switching probability is *smaller* early in the tracking process and *larger* in the latter part of the tracking process. Because of this unexpected response, the same analysis for the IMM algorithm switching processes is also conducted on the actual data in TBM profile 5.

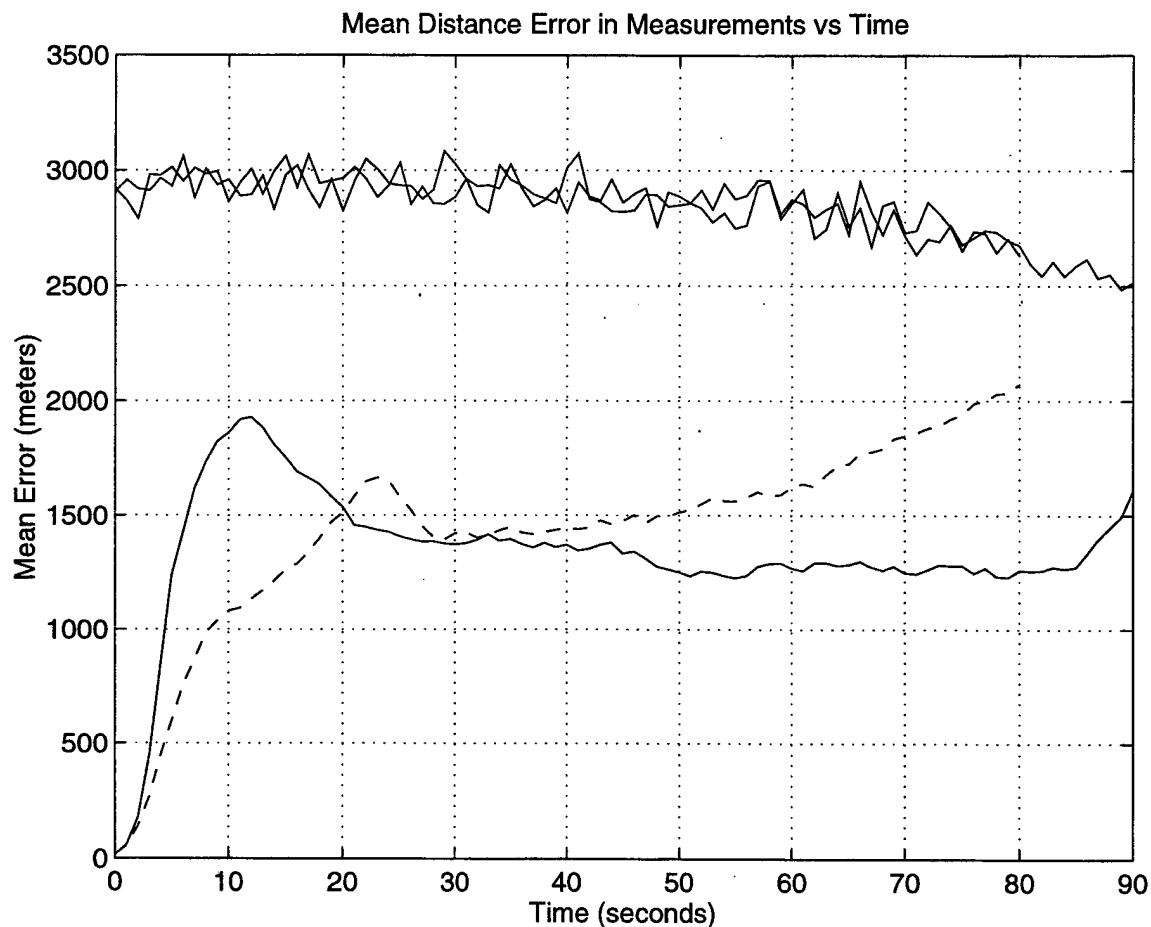


Figure 6.10(c) IMM with Sigmoid Switching Process vs. IMM with Constant Switching Probability, $\rho_{11}=0.75$ (500 Runs).

D. TBM PROFILE 5

The final TBM data chosen to be highlighted in this study is the TBM trajectory of profile 5. As in the two previous examples, the α - β - γ , EKF and IMM tracking algorithms are implemented on the TBM trajectory. Figure 6.11 shows a plot of the TBM trajectory of profile 5.

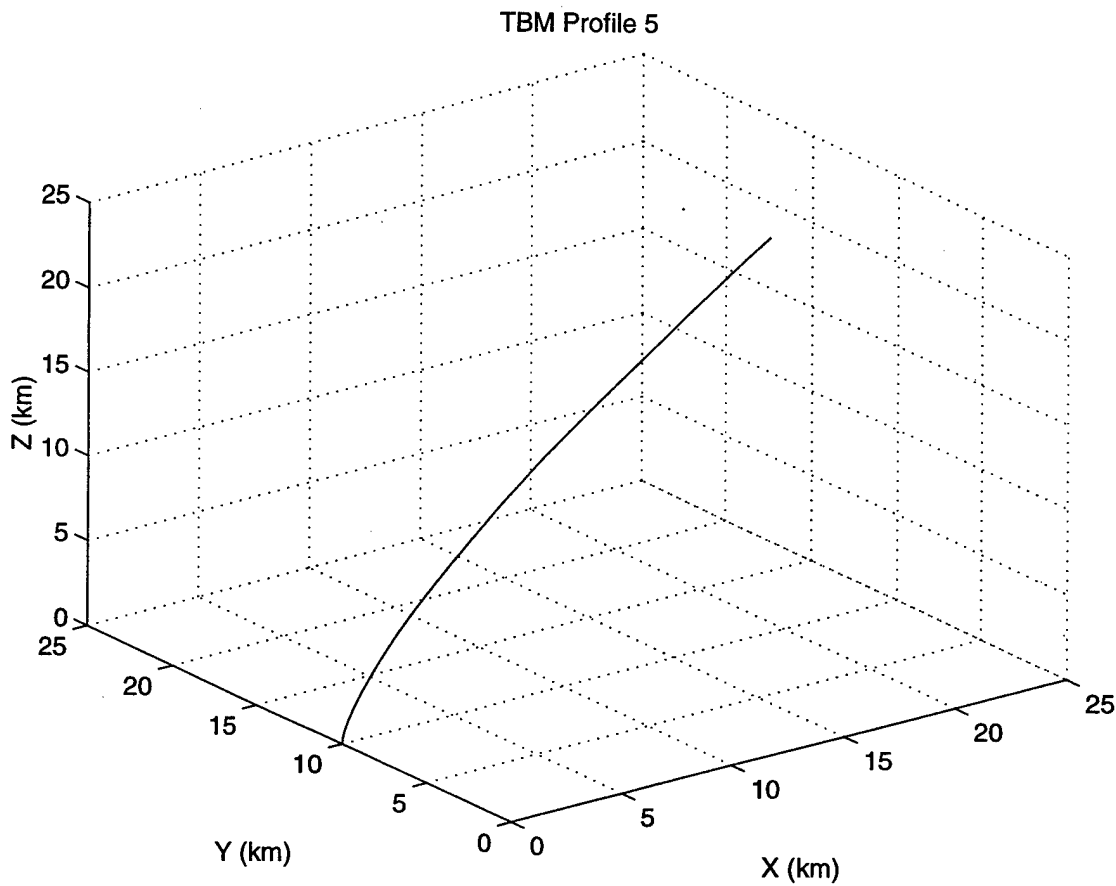


Figure 6.11 TBM Trajectory (Profile 5).

1. Alpha-Beta-Gamma Tracker Results

Figure 6.12(a) shows a plot of the TBM profile 5 with added measurement noise. The result of the α - β - γ tracking algorithm is shown in Figure 6.12(b) with the filtered trajectory superimposed on the TBM trajectory for profile 5. These results are obtained over 100 simulation runs, with $\alpha=0.6$.

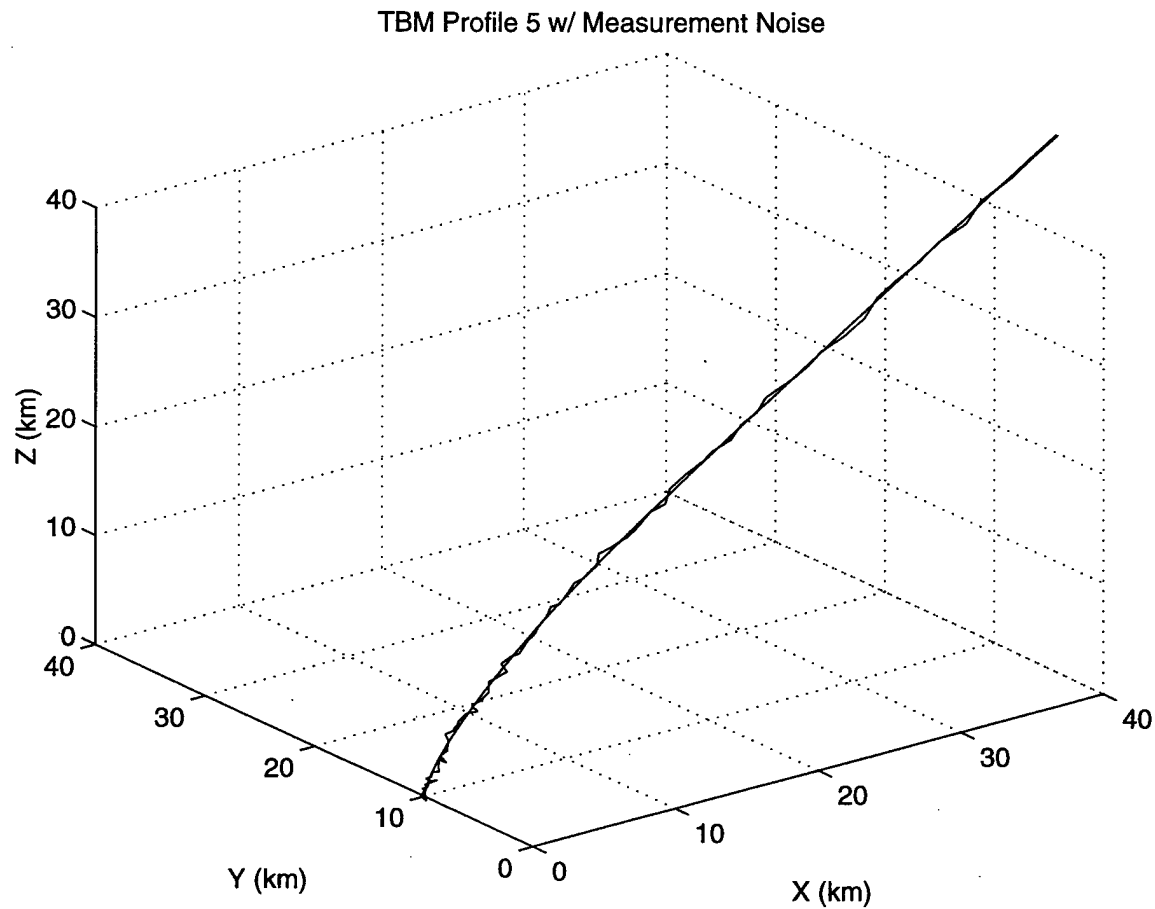


Figure 6.12(a) TBM Trajectory (Profile 5) with Measurement Noise, 100 Runs.

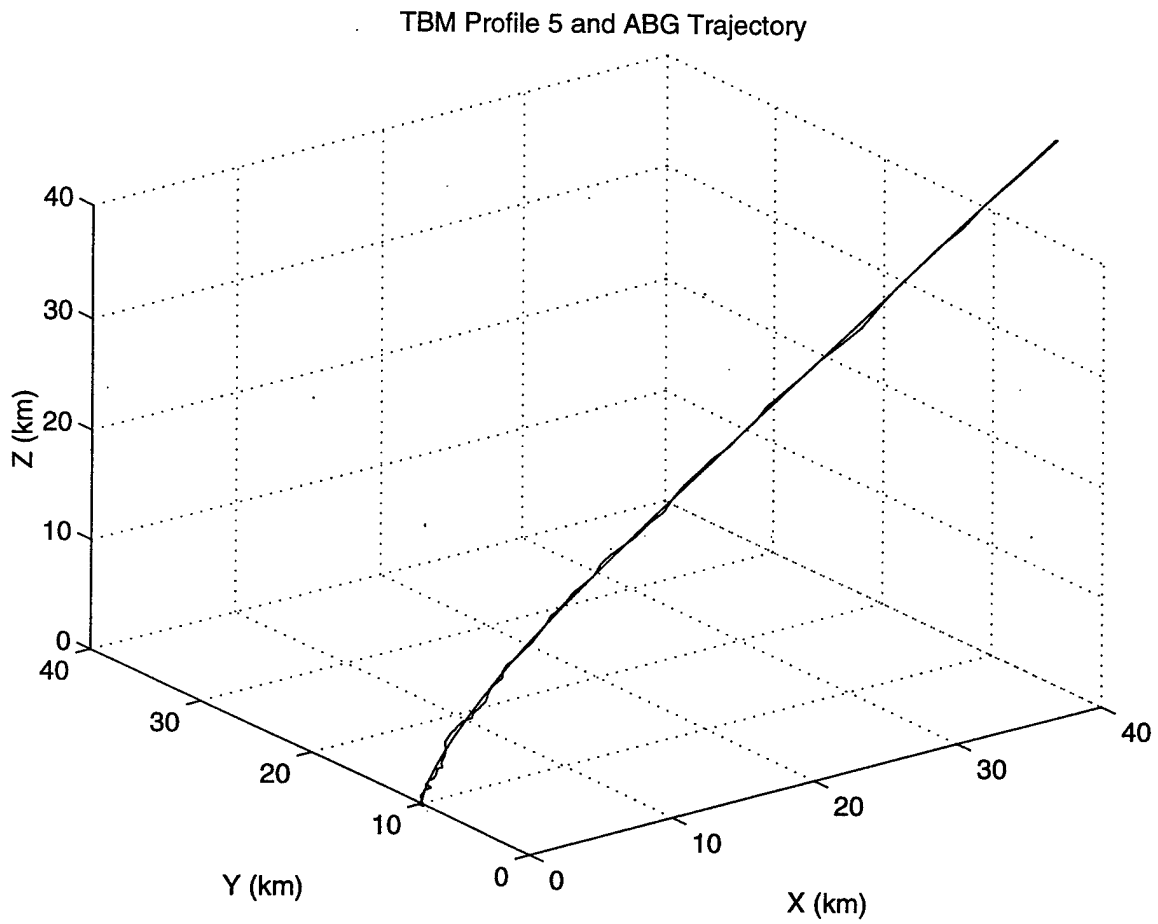


Figure 6.12(b) TBM Trajectory (Profile 5) and α - β - γ Trajectory, $\alpha=0.6$, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.12(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot, shown with the large spike, is the mean distance error using the α - β - γ tracking algorithm. These results indicate that the α - β - γ tracker reduces the mean measurement noise by approximately 30 percent despite a large transient error which is present in the first 10 seconds of the filter. This error is shown in Figure 6.12(c) as a spike that peaks to approximately 9,500 meters.

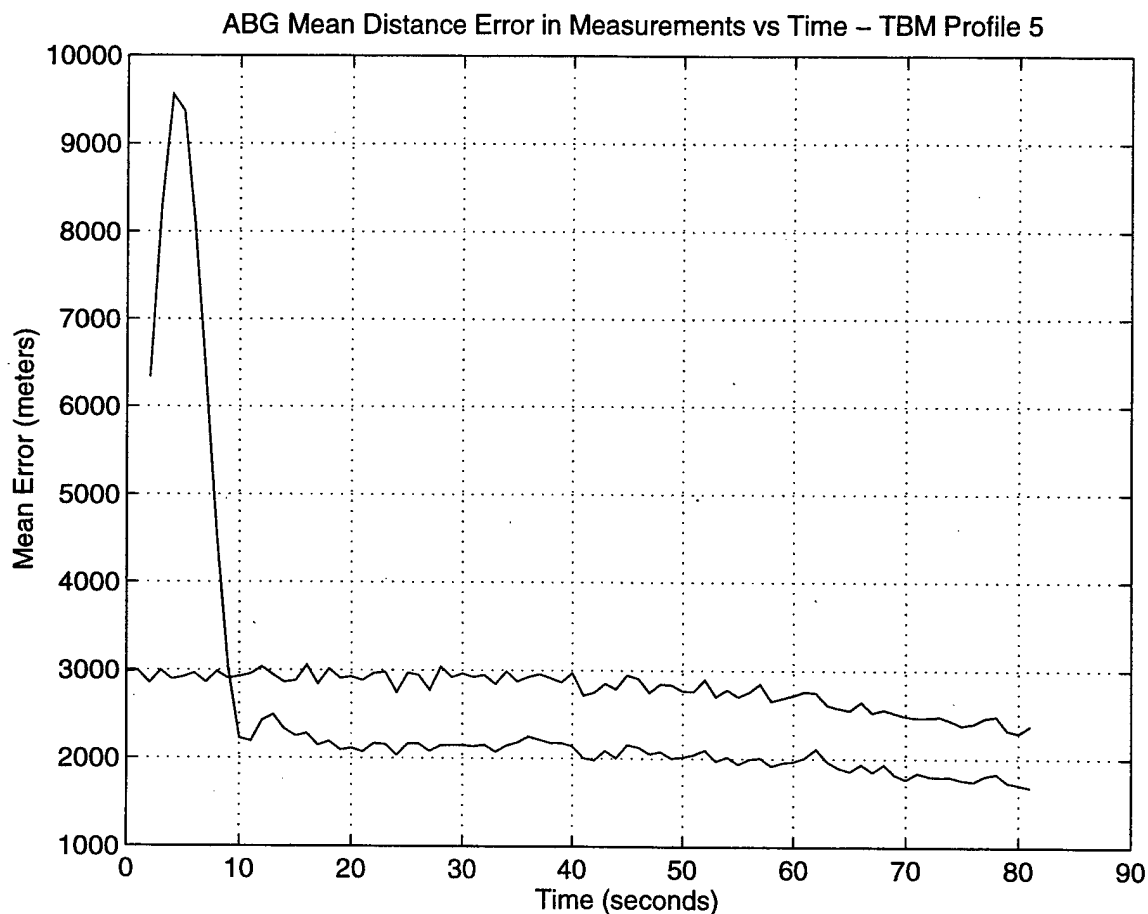


Figure 6.12(c) α - β - γ Tracker (Profile 4) Mean Distance Error, $\alpha=0.6$, 500 Runs.

2. EKF (Accelerating Model) Results

Figure 6.13(a) shows the TBM trajectory for profile 5 with added measurement noise. The result of the EKF (accelerating model) algorithm is shown in Figure 6.13(b) with the filtered trajectory superimposed on the TBM trajectory for profile 5. These results are obtained over 100 simulation runs, with $q^2=10$.

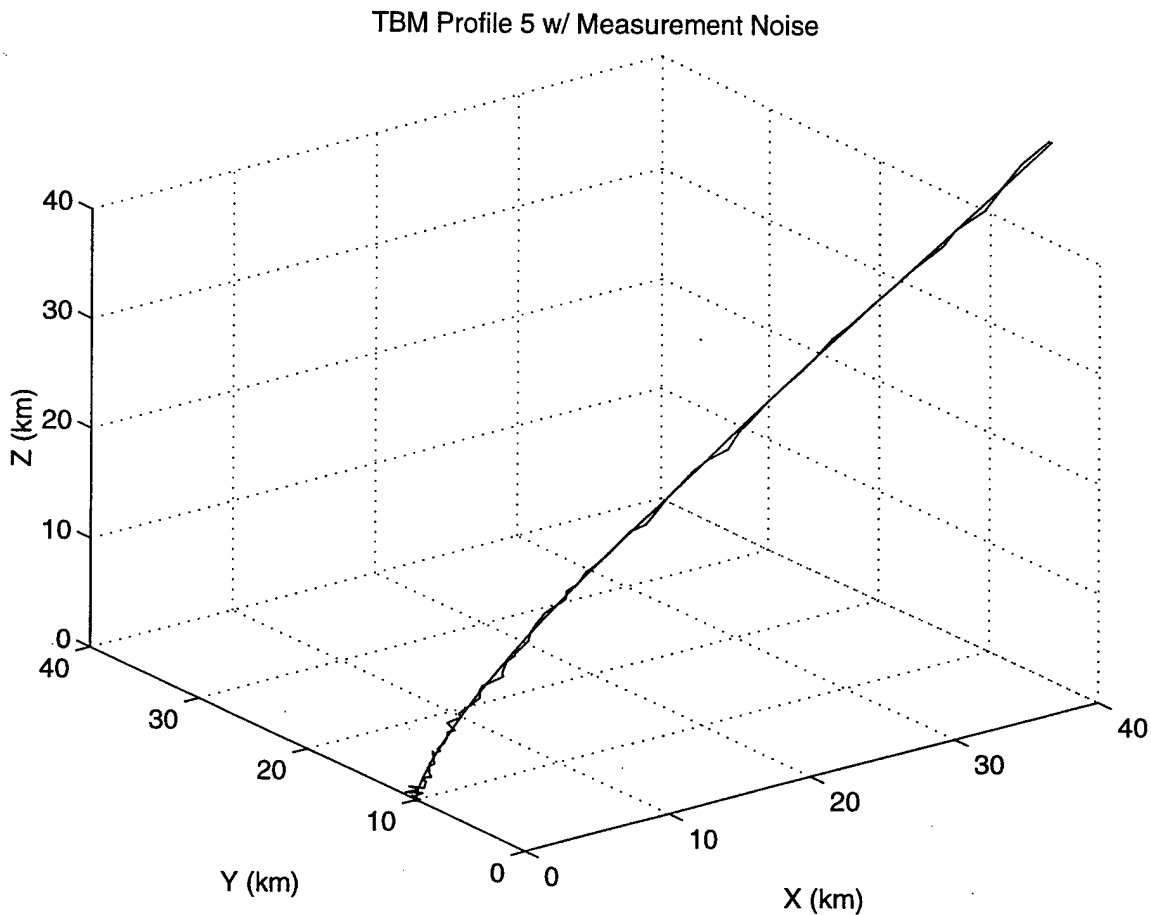


Figure 6.13(a) TBM Trajectory (Profile 5) with Measurement Noise, 100 Runs.

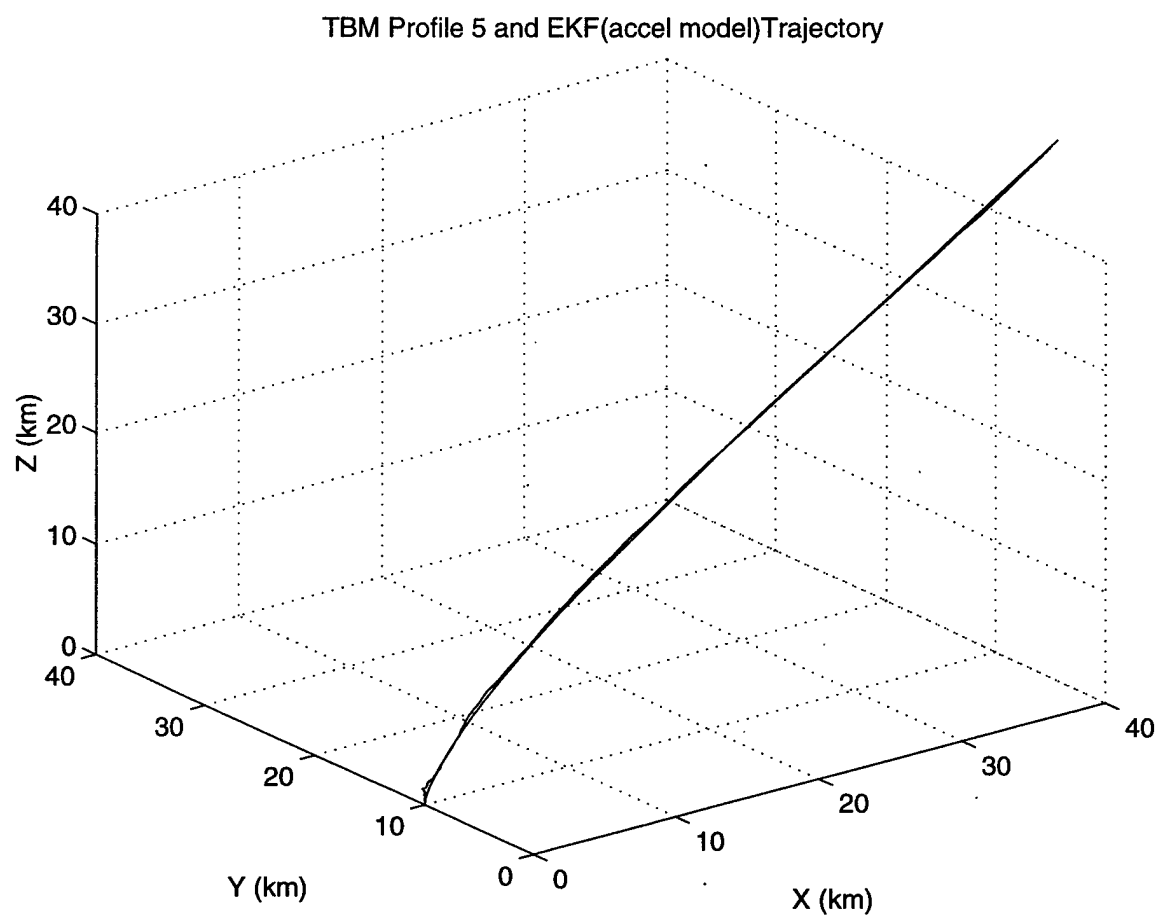


Figure 6.13(b) TBM Trajectory (Profile 5) and EKF Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.13(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the EKF tracking algorithm. These results indicate that the EKF algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1900 meters.

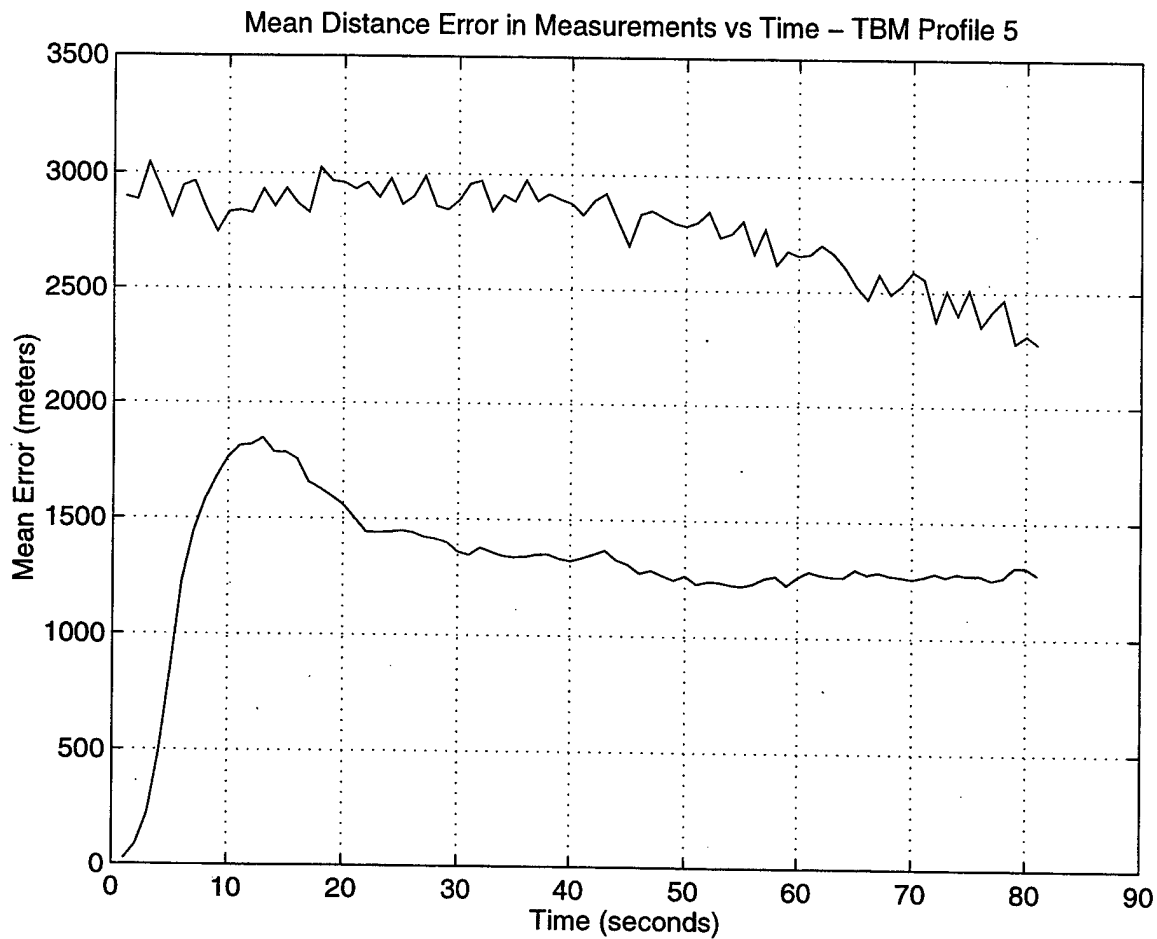


Figure 6.13(c) EKF (Profile 5) Mean Distance Error, 500 Runs.

3. IMM Results

Figure 6.14(a) shows the TBM trajectory for profile 5 with added measurement noise. The result of the IMM algorithm is shown in Figure 6.14(b) with the filtered trajectory superimposed on the TBM trajectory for profile 5. These results are obtained over 100 simulation runs, with $q^2=10$. The switching process is modeled using a sigmoid function that switches element ρ_{11} from a value of 1.0 to 0.5. The altitude at the maximum burn time in this profile is approximately 44 km, and in this model the IMM algorithm is set to start anticipating a change between models after the missile reaches an altitude of 39 km.

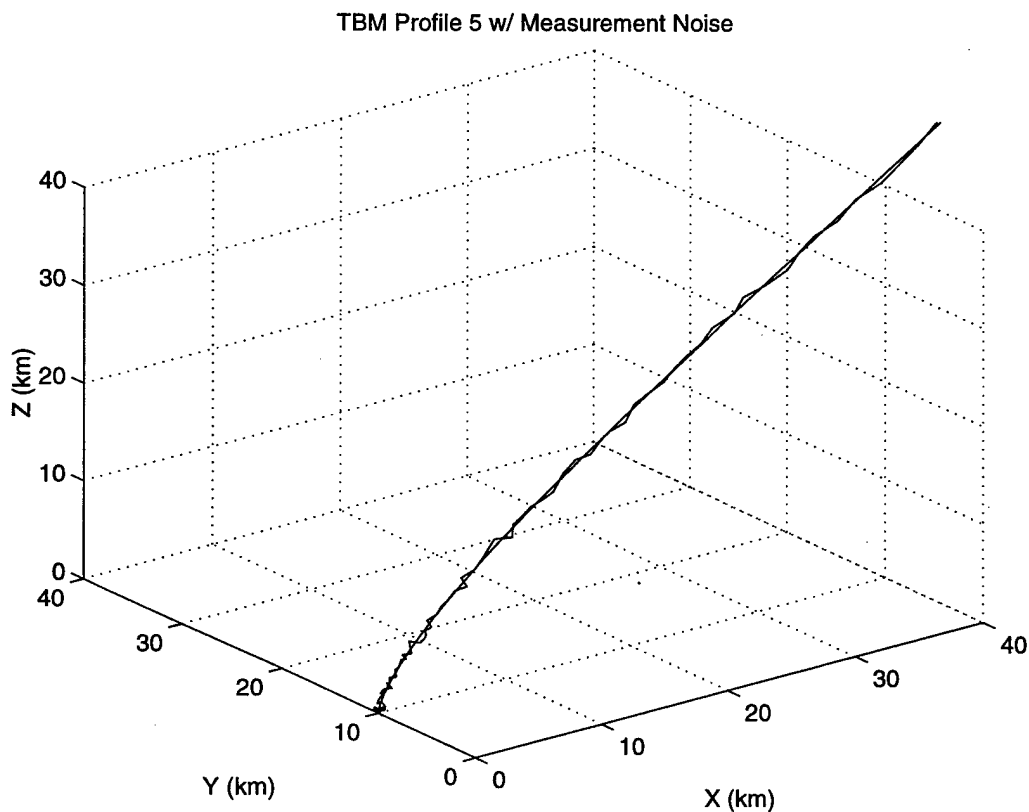


Figure 6.14(a) TBM Trajectory (Profile 5) with Measurement Noise, 100 Runs.

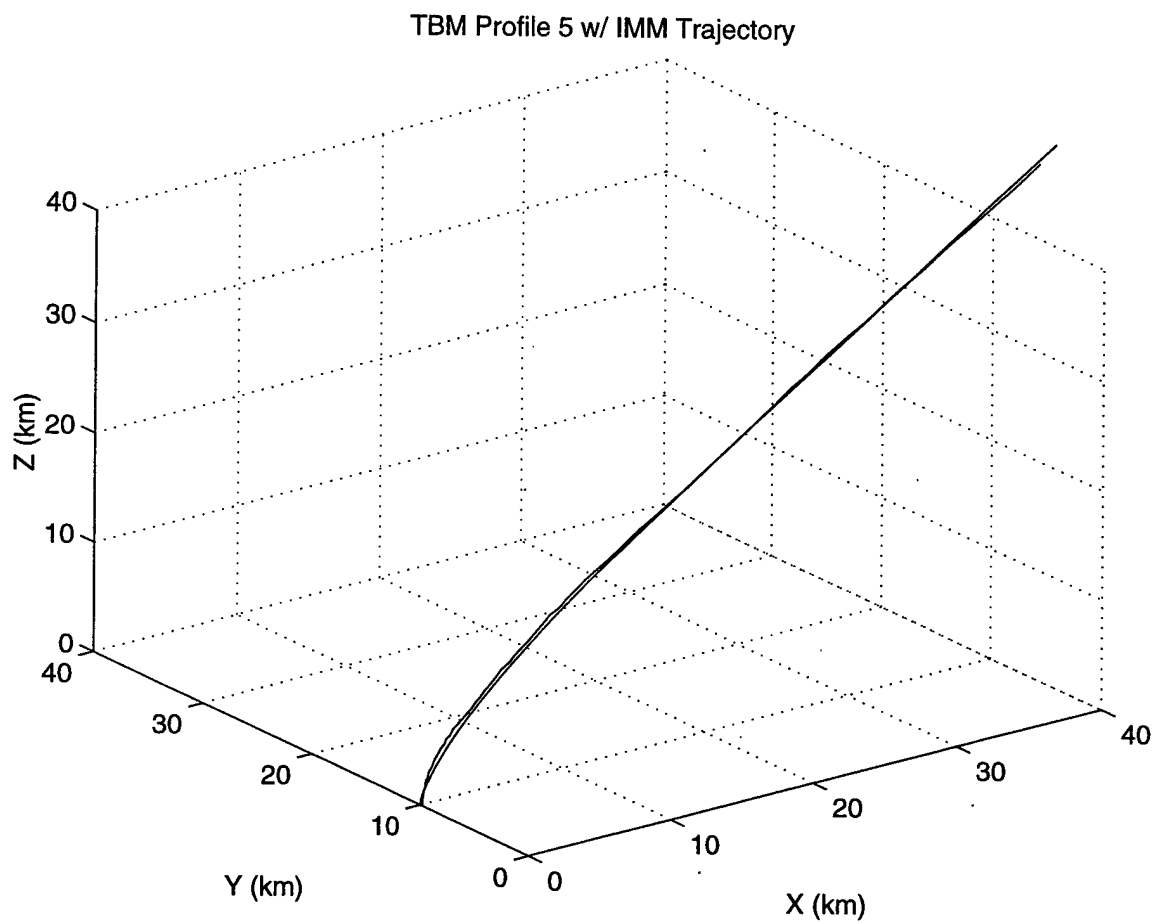


Figure 6.14(b) TBM Trajectory (Profile 5) and IMM Trajectory, 100 Runs.

The mean distance error in measurements is calculated over 500 simulation runs, and is shown in Figure 6.14(c). The upper plot is the mean measurement noise that is observed by the sensor platform, and the lower plot is the mean distance error using the IMM tracking algorithm. These results indicate that the IMM algorithm reduces the mean measurement noise by approximately 50 percent with an initial peak error of approximately 1500 meters. The rise in the mean distance error in the last few seconds indicates the IMM filter is anticipating the switch to the ballistic model.

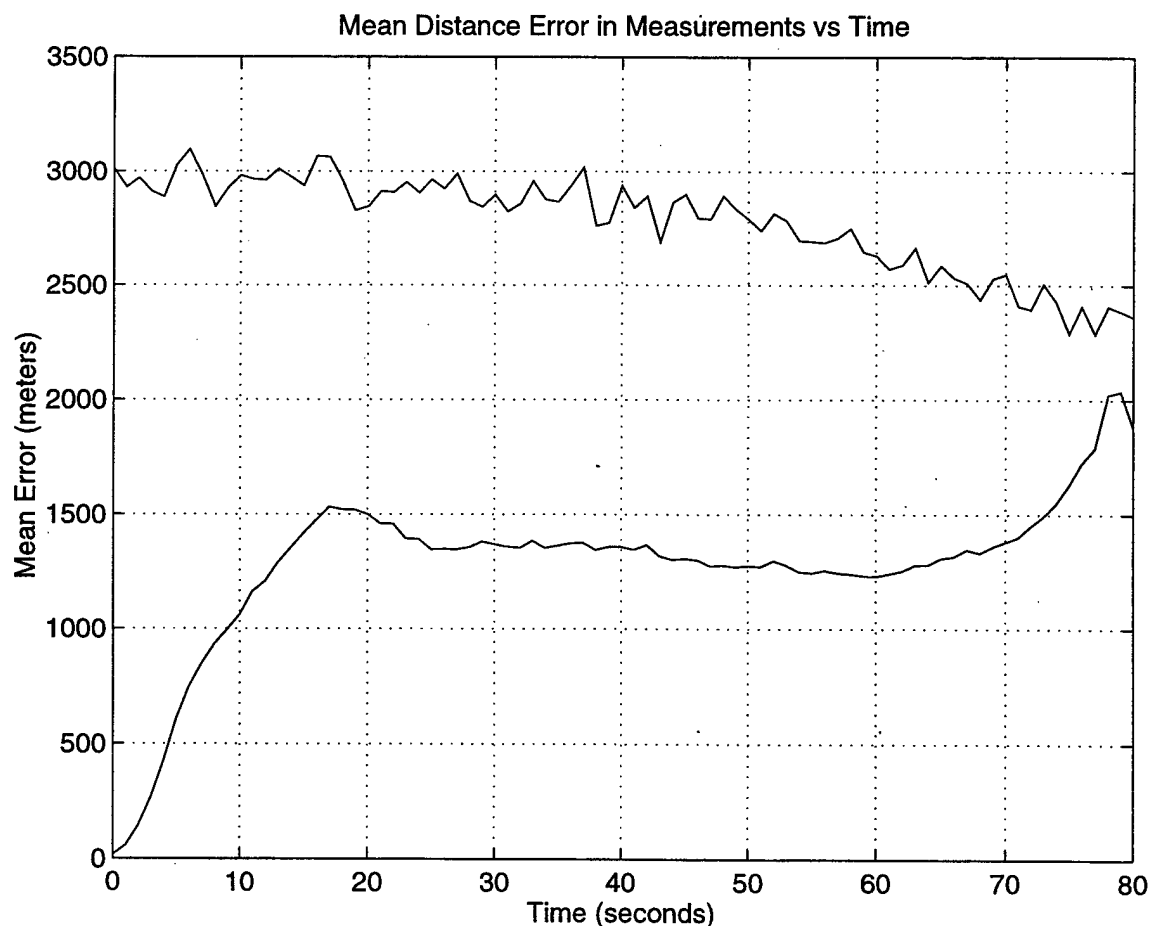


Figure 6.14(c) IMM (Profile 5) Mean Distance Error, 500 Runs.

4. Comparison of Mean Distance Error

Figure 6.15(a) shows a comparison of the mean distance error plots for the α - β - γ tracker (shown as a dash-dot line), the EKF accelerating model (shown as a dashed line), and the IMM algorithm (shown as a solid line). Figure 6.15(b) shows a close-up of the comparison. As expected, the results of the EKF and the IMM are similar, since the IMM algorithm does not switch to the ballistic model.

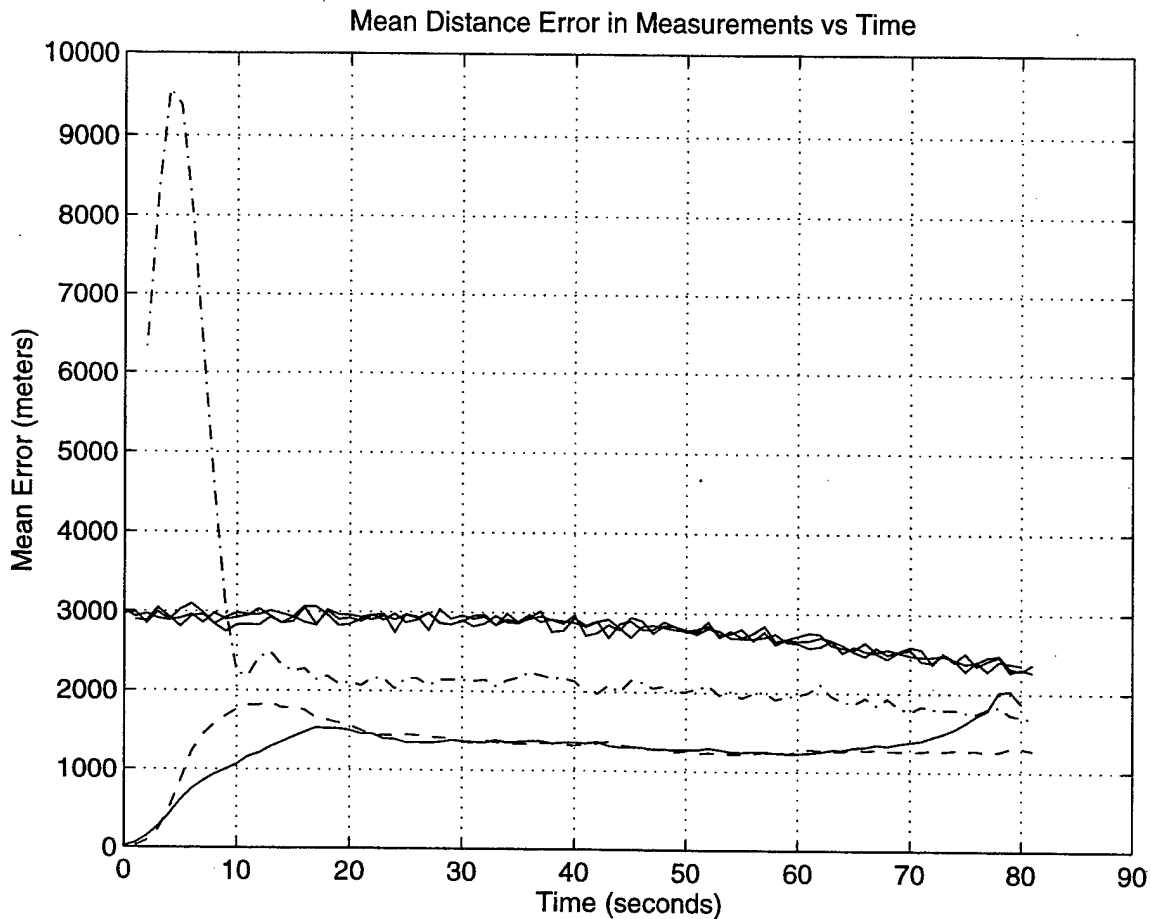


Figure 6.15(a) Comparison of α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

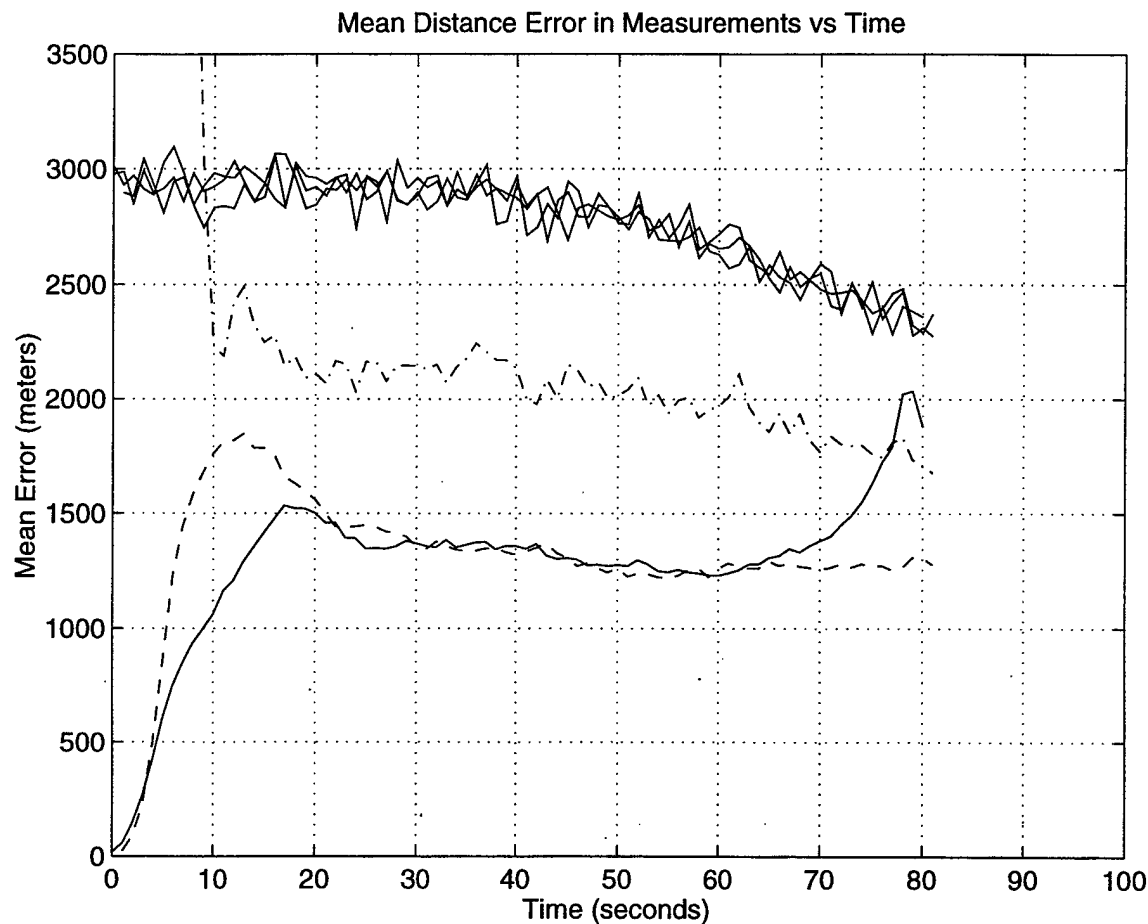


Figure 6.15(b) Comparison (Close-up) of Mean Distance Error, 500 Runs.

An analysis of the IMM algorithm switching process is conducted for TBM profile 5 using a constant value for the switching probability, with $p_{11}=0.75$. The results are compared to the previous example of the IMM algorithm, which uses a sigmoid switching process. Figure 6.15(c) shows a comparison of the mean distance error for the IMM algorithm utilizing a sigmoid switching function (shown as a solid line) and the IMM algorithm utilizing a constant switching probability (shown as a dashed line). The plots of the mean distance error for each switching process behave as expected, with

results similar to the results obtained from both the simulated data and the actual data from TBM profile 1. The familiar trade-off in performance is noted, with the IMM algorithm utilizing the sigmoid switching function performing better in the early part of the tracking process, and the IMM algorithm utilizing the constant switching probability performing better in the latter part of the tracking process.

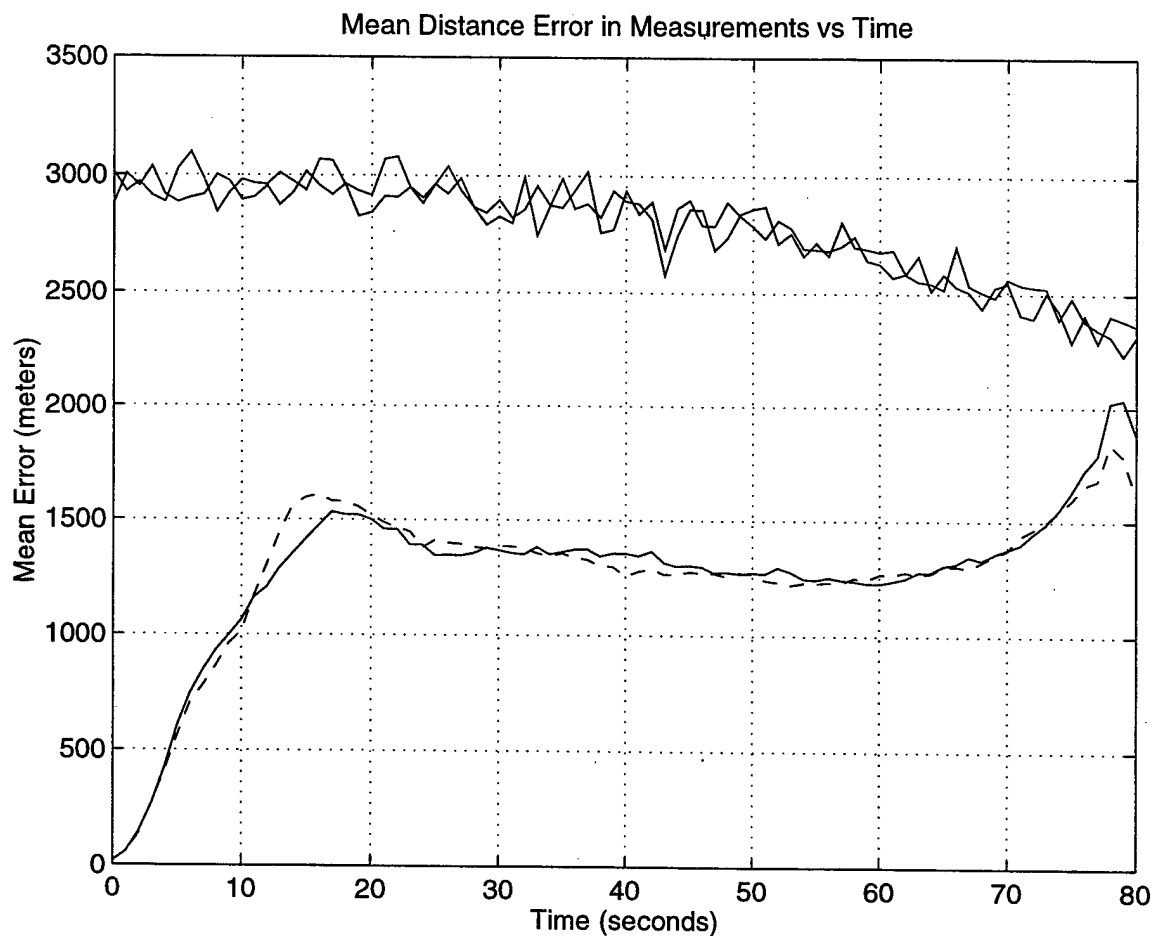


Figure 6.15(c) IMM with Sigmoid Switching Process vs. IMM with Constant Switching Probability, $\rho_{11}=0.75$ (500 Runs).

E. COMPARISON OF TBM PROFILES

A comparison of the tracking quality of the different algorithms is shown in Figures 6.16 through 6.20. TBM profiles 1, 4 and 5 were initially chosen for analysis in this section because of their different burn times listed in the TBM profile data. It was thought that the differences in burn times might identify some differences in the performance of the algorithms. However, the graphs show near identical results with only subtle differences in the performance of each individual missile characteristic. By reducing the mean distance error by approximately 50 percent, the IMM algorithm proved to be the most accurate tracker of the three filtering algorithms.

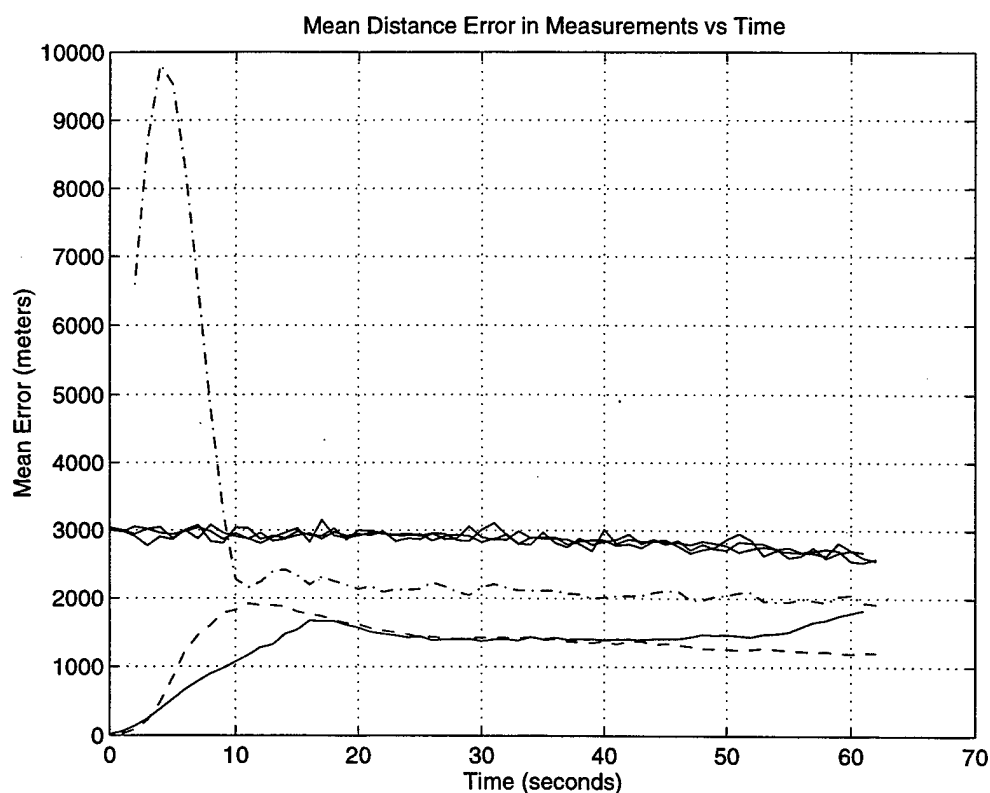


Figure 6.16 Profile 1, α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

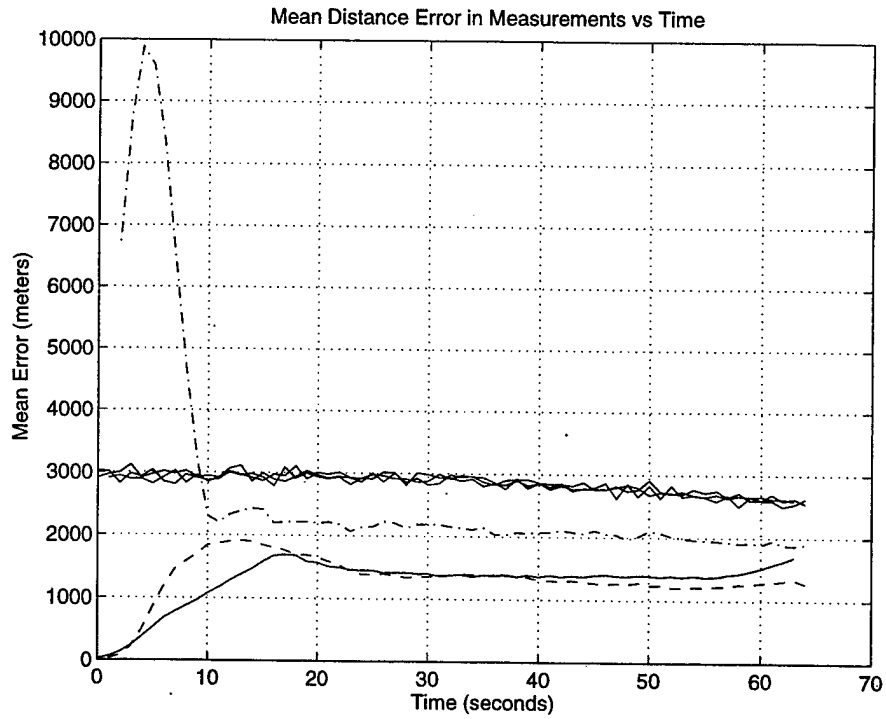


Figure 6.17 Profile 2, α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

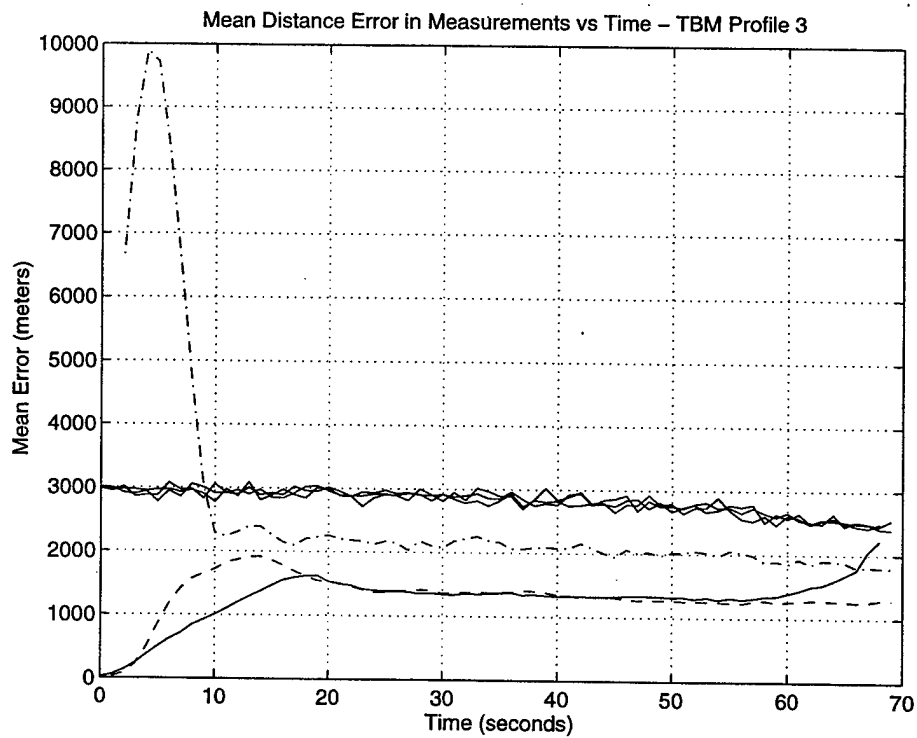


Figure 6.18 Profile 3, α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

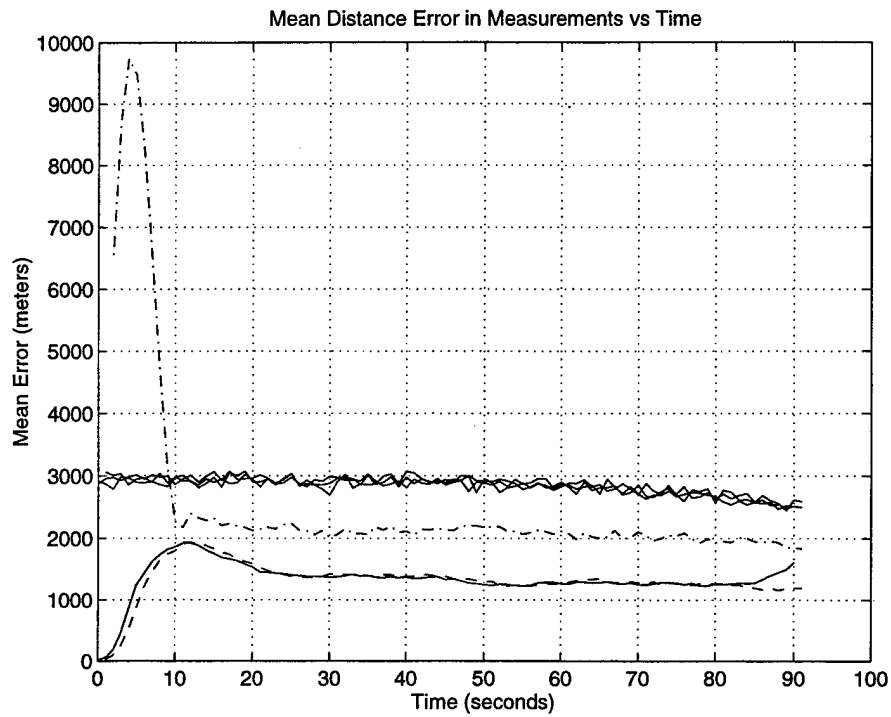


Figure 6.19 Profile 4, α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

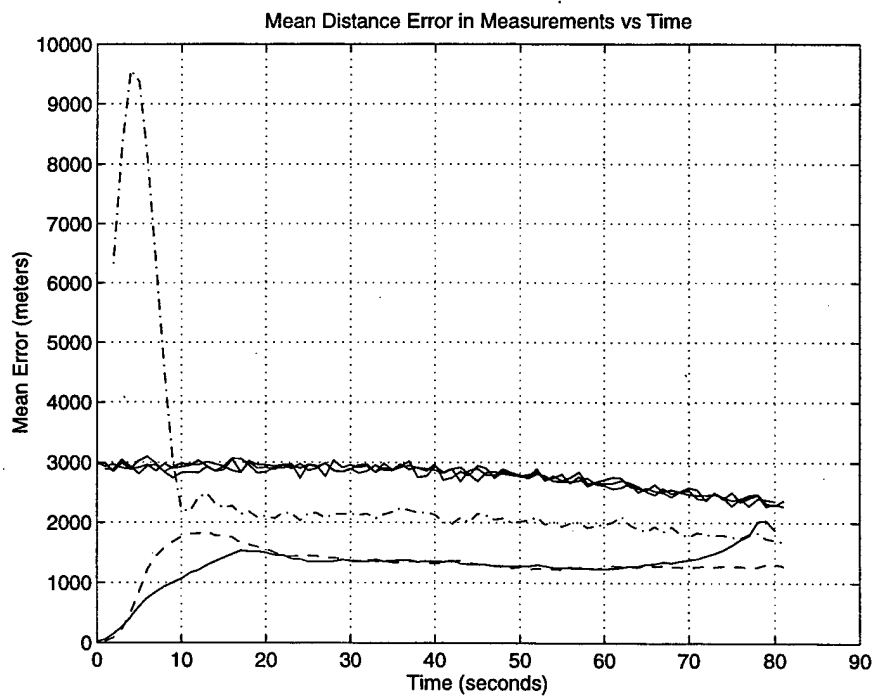


Figure 6.20 Profile 5, α - β - γ , EKF and IMM Mean Dist. Error, 500 Runs.

The tracking results of the actual TBM data in this chapter are consistent with the tracking results of the simulated TBM data in Chapters II through V. In all of the TBM profiles, the mean measurement noise observed by the sensor is reduced by approximately 50 percent with a mean distance error of approximately 1500 meters. Because the information provided in TBM profiles 1 through 5 only contains data up to the maximum burn time, the missile's transition area and post booster cut-off areas are not studied. The next chapter provides a summary of the analysis of TBM tracking during boost phase. Conclusions are made and recommendations for follow-on studies are presented.

VII. CONCLUSION

The tracking of TBMs during their boost phase has been investigated, and α - β - γ , EKF and IMM tracking algorithms have been developed. The IMM tracking algorithm was shown to be the most effective algorithm for tracking TBMs during boost and transition phases. As shown in Chapter IV, the α - β - γ tracker performed only slightly better than the mean measurement noise observed by the sensor. Additionally, large transient errors were present in the initial few seconds of tracking. The EKF algorithm (accelerating model), shown in Chapters III and V, encountered significant difficulty tracking TBMs after booster cut-off. As a result, large peaks were present in the transition area of the mean distance error plots for the EKF algorithm. In this study, the IMM algorithm was shown to be the best overall tracking algorithm because of its ability to track TBMs during the large initial accelerations encountered during boost phase, and during the change in missile dynamics encountered in the TBM's transition to a ballistic phase.

In the analysis of both the simulated and actual TBM data, the IMM algorithm outperformed all other tracking algorithms. In the simulated data, the IMM tracking algorithm significantly reduced the mean measurement noise observed by the sensor by approximately 75 percent (with a mean distance error of approximately 400 meters). In addition, the mean distance error during the missile's transition phase (after booster cut-off) was significantly reduced to approximately 200 meters. In the actual TBM data, the IMM tracking algorithm consistently reduced the mean measurement noise observed by

the sensor by approximately 50 percent (with a mean distance error of approximately 1500 meters). The difference in IMM performance between the simulated data and the actual data was attributed to two factors. First, in the simulated data, the TBM launch point was intentionally offset from the origin and the resulting distance between the sensor and the launch point was approximately 92 km. In the actual TBM data, the downrange distance and altitude were referenced to the launch point, and for plotting purposes, the TBM was launched from the origin. A larger distance of 141 km resulted between the sensor and the launch point. Secondly, the sampling interval in the simulated data was set at 0.1 seconds while in the actual data, the interval was 1 second. This longer time interval between missile position measurements, combined with the increased distance between the sensor and missile launch point led to larger distance errors in the actual data.

Follow-on studies should concentrate on the analysis of additional TBM profile data from actual missile launches to include data over the entire trajectory. This will allow for further investigation of the IMM algorithm in the transition areas of actual TBM data.

APPENDIX A. SOURCE CODE FOR BALLISTIC MISSILE SIMULATION

A. MATLAB® CODE FOR INITIALIZATION

The following is the MATLAB® program used to initialize the ballistic missile simulation.

```
%*****
% PtMissileInitS.m
% LT Tony San Jose
% Thesis Advisor: R.G Hutchins
% 03FEB98
%
% This script file initializes the flat earth point
% missile simulation
%*****

% define globals
global g mass T tToff troll cfri xinit tmax sinterval;

g = 9.8;           % gravity, meters/sec^2
T = 6*g;           % missile acceleration
tToff = 60; %100;   % time of thrust shut off (seconds)
troll = 20; %30;    % time of missile rollover(seconds)
cfri = 0.05;       % coefficient of friction
sinterval = 0.1;   % sampling interval (seconds)

tmax = 520;        % max simulation time (seconds)

wel = (40*(pi/180))/(tToff - troll); % rotation in elev (rads/sec)
waz = (15*(pi/180))/(tToff - troll); % rotation in azimuth (rads/sec)

minstep = 1e-5;    % minimum step size
numsamp = tmax/minstep; % number of samples

xinit = [30 * 1000; % Initial Missile x position (m)
         0;         % Initial Missile x velocity (m/s)
         0;         % Initial Missile x acceleration (m/s^2);
         40 * 1000; % Initial Missile y position (m)
         0;         % Initial Missile y velocity (m/s)
         0;         % Initial Missile y acceleration (m/s^2);
         0;         % Initial Missile z position (m)
         0.001;     % Initial Missile z velocity (m/s)
         0];        % Initial Missile z acceleration (m/s^2)
```

B. MATLAB® CODE FOR MISSILE DYNAMICS FUNCTION

The following is the MATLAB® program used to generate missile dynamics using flat earth equations of motion.

```
function xdot = FlatEarthPtEqns(u)
%*****
% This Function computes the Flat Earth, Point Mass Equations
% for Missile Dynamics.
%
% LT Tony San Jose
% Thesis Advisor: R.G Hutchins
% 03FEB98
%
% The input vector is defined as:
%     u(1) = T,          thrust along the missile velocity vector
%     u(2) = we,        Velocity Vector Rotation Rate in elevation
%     u(3) = waz,        Velocity Vector Rotation Rate in azimuth
%
% The State Vector is defined as:
%     Position Variables
%     u(4) = Px,  Position North of (0,0,0)
%     u(7) = Py,  Position East of (0,0,0)
%     u(10)= Pz,  Height
%
%     Position Velocities
%     u(5) = U,   D(Px)/dt
%     u(8) = V,   D(Py)/dt
%     u(11)= W,   D(Pz)/dt
%
%     Position Accelerations
%     u(6) = Ax,  D(Px)/dt
%     u(9) = Ay,  D(Py)/dt
%     u(12)= Az,  D(Pz)/dt
%
% Related Quantities
%     g,          Gravitational Force = 9.8 meters/sec^2
%     cfric        coefficient of friction
%     rho,         air density with altitude
%     mass,        missile mass
%     tToff,       Time of Thrust Shutoff
%     troll,       Time of Missile Rollover < tToff
%
%*****
% Declare Global Variables
%     global g mass tToff troll cfric tmax;
%
% Define Control Variables from Inputs
%     T = u(1); % thrust along missile velocity
%     we1 = u(2); % turn rate in elevation
```

```

    waz = u(3); % turn rate in azimuth

% Define State Variables from Inputs
    x = u(4:12);
% Location Variables
    Px = x(1); % Position in Direction of North Pole
    Py = x(4); % Position At Equator in y
    Pz = x(7); % Position At Equator in z

% Body-Axes Velocities
    U = x(2); % velocity in Px direction
    V = x(5); % velocity in Py direction
    W = x(8); % velocity in Pz direction ("Up")

% Speed, Atmospheric Density and Drag
    Vxy2 = U^2 + V^2;
    Vxy = sqrt(Vxy2);
    Vxz2 = U^2 + W^2;
    Vt2 = Vxz2 + V^2;
    Vt = sqrt(Vt2);
    az = atan2(V,U);
    el = atan2(W,Vxy);

% Atmospheric Density in kg/meter^3
    if Pz < 0 % Travel inside the Earth is Viscous
        rho = 10^2;
    elseif Pz < 9144 % Altitudes below 9144 meters
        rho = 1.22557*exp(-Pz/9144);
    else % Altitudes above 9144 meters
        rho = 1.75228763*exp(-Pz/6705.6);
    end

    beta = cfri*rho;
    Tacc = T/Vt;

% Compute the Derivatives
    dPx = U;
    dPy = V;
    dPz = W;

% Azimuth and Elevation Rollover
    dU = -waz*V + wel*W*cos(az) - beta*U + Tacc*U;
    dV = waz*U + wel*W*sin(az) - beta*V + Tacc*V;
    dW = -wel*Vxy - g - beta*W + Tacc*W;

xdot = [dPx;
        dU;
        0;
        dPy;
        dV;
        0;
        dPz;
        dW;
        0];

```

C. MATLAB® CODE FOR PLOTTING MISSILE SIMULATION

The following is the MATLAB® program used to plot the output of the
SIMULINK™ model, *FlatEPtMissileSim.m*

```
%*****  
% FlatEPtPlotsS.m  
%  
% This file plots the results of the SIMULINK missile simulation  
%*****  
  
% Define Variables  
t = missilevec(:,1);  
x = missilevec(:,2);  
vx = missilevec(:,3);  
ax = missilevec(:,4);  
y = missilevec(:,5);  
vy = missilevec(:,6);  
ay = missilevec(:,7);  
z = missilevec(:,8);  
vz = missilevec(:,9);  
az = missilevec(:,10);  
  
x_km = x/1000;  
y_km = y/1000;  
z_km = z/1000;  
  
sxy = vx.^2 + vy.^2;  
speed = sqrt(sxy + vz.^2);  
sxy = sqrt(sxy);  
dist = sqrt(x.^2 + y.^2);  
az = atan2(vy,vx)*180/pi;  
el = atan2(vz,sxy)*180/pi;  
xaccel = ax/9.8;  
yaccel = ay/9.8;  
zaccel = az/9.8;  
total_accel = sqrt(xaccel.^2 + yaccel.^2 + zaccel.^2);  
  
%*****  
% Plot Data  
%*****  
  
figure(1)  
plot(x_km,z_km,'r-');  
axis('equal'),grid;  
xlabel('X (km)'),ylabel('Z (km)');  
title('Missile Z vs. X Plot');  
% print -deps ch2fg2a  
  
figure(2)  
plot(x_km,y_km,'r-');
```

```

    axis('equal');
    xlabel('X (km)'), ylabel('Y (km)'), grid;
    title('Missile Y vs. X Plot');
%   print -deps ch2fg2b

figure(3)
    plot(t,(dist/1000),'r-');
    ylabel('Down Range Dist (km)'), xlabel('Time (seconds)'), grid;
    title('Down Range Distance vs Time');
%   print -deps ch2fg2c

figure(4)
    plot(t,z_km,'r-');
    axis('equal');
    ylabel('Missile Altitude (km)'), xlabel('Time (seconds)'), grid;
    title('Missile Altitude vs Time (kilometers)');
%   print -deps ch2fg2d

figure(5)
    plot(t,speed,'r-');
    ylabel('Missile Speed (m/s)'), xlabel('Time (seconds)'), grid;
    title('Missile Speed vs Time');
%   print -deps ch2fg2e

figure(6)
    plot(t,az,'r-');
    title('Missile Azimuth Heading vs Time');
%   print -deps ch2fg2f

figure(7)
    plot(t,el,'r-');
    title('Missile Elevation Angle vs Time');
%print -deps ch2fg2g

figure(8)
    plot(dist,z,'r-');
    axis('equal');
    title('Down Range Distance vs Height');
%print -deps ch2fg2h

figure(9)
    plot3(x,y,z,'r-');
    axis('equal');
    ylabel('Y (m)'), xlabel('X (m)'), zlabel('Z (m)'), grid;
    title('Three Dimensional Missile Trajectory in meters');
%   print -deps ch2fg2i

figure(10)
    plot3(x_km,y_km,z_km,'r-');
    axis('equal');
    ylabel('Y (km)'), xlabel('X (km)'), zlabel('Z (km)'), grid;
    title('Three Dimensional Missile Trajectory in kilometers');
%   print -deps ch2fg2j

```

```

figure(11)
    plot3(x(1:1200),y(1:1200),z(1:1200),'r-');
    axis('equal');
    ylabel('Y (m)'), xlabel('X (m)'), zlabel('Z (m)'), grid;
    title('Missile Trajectory - Initial 120 Seconds in meters');
%    print -deps ch2fg2k

figure(12)
    plot3(x_km(1:1200),y_km(1:1200),z_km(1:1200),'r-');
    axis('equal');
    ylabel('Y (km)'), xlabel('X (km)'), zlabel('Z (km)'), grid;
    title('Missile Trajectory - Initial 120 Seconds in kilometers');
%    print -deps ch2fg2l

figure(13)
    plot(t,xaccel,'r-');
    ylabel('gs'), xlabel('Time (seconds)'), grid;
    title('Missile Acceleration in X vs Time');
%    print -deps ch2fg2m

figure(14)
    plot(t,yaccel,'r-');
    ylabel('gs'), xlabel('Time (seconds)'), grid;
    title('Missile Acceleration in Y vs Time');
%    print -deps ch2fg2n

figure(15)
    plot(t,zaccel,'r-');
    ylabel('gs'), xlabel('Time (seconds)'), grid;
    title('Missile Acceleration in Z vs Time');
%print -deps ch2fg2o

figure(16)
    plot(t,total_accel,'r-');
    ylabel('gs'), xlabel('Time (seconds)'), grid;
    title('Missile Acceleration vs Time');
%print -deps ch2fg2p

```

APPENDIX B. SOURCE CODE FOR EXTENDED KALMAN

FILTER TRACKING ALGORITHM

The following is the MATLAB[®] program used in the tracking of the ballistic missile base trajectory .

```
%*****
% efk.m
% LT Tony San Jose
% Thesis Advisor: R. G. Hutchins
% 03FEB98
%
% This program uses an EKF to filter the sensor measurement noise
% from the ballistic missile base trajectory developed using
% SIMULINK Random noise is added in the sensor
% measurement process. Actual missile track is generated in
% FlatEarthMissile SIMULINK model.
%
% INPUT
% missilevec:      state vector = [x,Vx,Ax,y,Vy,Ay,z,Vz,Az]'
%
% OUTPUT
% mean_K_track      Kalman estimated positions
%*****

% Load simulation workspace
clear all
load dat1;
missilevec = missilevec';

% Define the number of simulation loops
nloops = 10;

% Define the sampling interval
delta = .1;

% Define the number of samples
nsamples = 1200;

% Initialize sensor data
Sensor_posit = [ 100 * 1000;      % sensor is 100 km in x
                 100 * 1000;      % sensor is 100 km in y
                 0 * 1000];      % sensor is 100 km in z

sigma_r = 10;      % 10 meters std dev in range
sigma_b = 1*pi/180; % 1 degree std dev in azimuth
sigma_e = 1*pi/180; % 1 degree std dev in elevation

R = diag([sigma_r^2,      % covariance matrix for uncorrelated
```

```

        sigma_b^2,      % range and bearing measurements
        sigma_e^2]);

% Define F matrix (TRANSITION MATRIX) for discrete time
% target motion,  $x(k+1) = F(k)x(k) + G$ 

    f_sub = [1, delta, (delta^2)/2;
              0, 1,      delta;
              0, 0,      1 ];

    F = [ f_sub, zeros(3), zeros(3);
          zeros(3), f_sub, zeros(3);
          zeros(3), zeros(3), f_sub ];

% Define G matrix
    G = -g * [0;
              0;
              0;
              0;
              0;
              0;
              (delta^2)/2;
              delta;
              0];

% Define the H matrix (MEASUREMENT MATRIX), assuming that the
% x, y, and z missile positions are observed directly;  $z(k) = H(k)x(k)$ 

    H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
          0, 0, 0, 1, 0, 0, 0, 0, 0;
          0, 0, 0, 0, 0, 0, 1, 0, 0];

% Initialize Q, the covariance of the plant noise
%  $q^2$  = scale factor to system noise covariance matrix Q,
% used to account for unmodeled target maneuver acceleration.

    q_sqr = 10;

    Q_sub = [ (delta^5)/20, (delta^4)/8, (delta^3)/6;
              (delta^4)/8,  (delta^3)/3, (delta^2)/2;
              (delta^3)/6,  (delta^2)/2,  delta ];

    Q = q_sqr * [ Q_sub, zeros(3), zeros(3);
                  zeros(3), Q_sub, zeros(3);
                  zeros(3), zeros(3), Q_sub ];

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
    for kk = 1: nloops
tic
kk

```



```

% define empty output matrices

% measurement positions (cartesian) w/error
zout_true_n = [];

% distance error between measurement and true position
error_true = [];

% Kalman estimated trajectory
K_track = [];
K_accel = [];

% error between Kalman track and actual track
track_error = [];

%*****
% This block is used for the initialization process. Initialize
% from a SWAG.
%*****
x_swag = missilevec(2:10,1);
x_swag(9) = 6*g;
p_swag = eye(9) * 10^4;

x_corr = x_swag;
P_corr = p_swag;

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
for ii = 2:nsamples

    % Process the measurement from Sensor

    % True missile position
    ztrue = [missilevec(2,ii);
             missilevec(5,ii);
             missilevec(8,ii)];

    %*****
    % convert current position to polar coordinates and add
    % sensor noise to the position, generating a noisy measurement
    % from the sensor.
    %*****

    % position relative to the sensor
    zrel = ztrue - Sensor_posit;
    % range from sensor
    r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2);
    % bearing from sensor
    b = atan2(zrel(2), zrel(1));
    % range in x/y plane
    r_prime = sqrt(zrel(1)^2 + zrel(2)^2);

```

```

% elevation from sensor
e = atan2(zrel(3), r_prime);
% add noise to the measurement
r_n = r + sigma_r * randn;
b_n = b + sigma_b * randn;
e_n = e + sigma_e * randn;

% measurement in polar + noise
z_polar_n = [r_n;
             b_n;
             e_n];

% measurement in cartesian coordinates + noise
z_cart_true_n = [r_prime*cos(b_n);
                 r_prime*sin(b_n);
                 r_n*sin(e_n) ] + Sensor_posit;

z_cart_rel_n = [r_prime*cos(b_n);
                r_prime*sin(b_n);
                r_n*sin(e_n) ];

% compute measurement error in cartesian coordinates
zdiff = ztrue - z_cart_true_n;
distorerror = sqrt(zdiff'*zdiff);

% Update the measurement array
% true cartesian measurement + error
zout_true_n = [zout_true_n, z_cart_true_n];

% measurement error (between true measurements)
error_true = [error_true, disterror];

%*****
% Prediction
%*****

% Kalman Filter prediction equations
x_predict = F * x_corr + G;
P_predict = F * P_corr * F' + Q;

%*****
% Correction
%*****

% Convert to relative position to compute RBE coordinates
x_1 = x_predict(1) - Sensor_posit(1);
x_4 = x_predict(4) - Sensor_posit(2);
x_7 = x_predict(7) - Sensor_posit(3);

% Convert prediction to Range, Bearing, Elevation
coordinates
r_hat = sqrt(x_1^2 + x_4^2 + x_7^2);
b_hat = atan2(x_4, x_1);

```

```

        e_hat = atan2(x_7, sqrt(x_1^2 + x_4^2));

% Determine expected measurement
        z_polar_hat = [r_hat;
                        b_hat;
                        e_hat];

% Observed minus expected measurements
        z_tilde = z_polar_n - z_polar_hat;

% The gradient of H evaluated at the most recent estimate
Hk_r2c1 = -x_4/(x_1^2 + x_4^2);
Hk_r2c4 = x_1/(x_1^2 + x_4^2);
Hk_r3c1 = (-x_1*x_7)/(sqrt(x_1^2 + x_4^2)*(x_1^2 + x_4^2 + x_7^2));
Hk_r3c4 = (-x_4*x_7)/(sqrt(x_1^2 + x_4^2)*(x_1^2 + x_4^2 + x_7^2));
Hk_r3c7 = (sqrt(x_1^2 + x_4^2))/(x_1^2 + x_4^2 + x_7^2);

% Determine H matrix
Hk = [x_1/r_hat, 0, 0, x_4/r_hat, 0, 0, x_7/r_hat, 0, 0;
      Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
      Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

% Compute Kalman Gain
K = P_predict * Hk' * inv(Hk * P_predict * Hk' + R);

% Correction equations
x_corr = x_predict + K * z_tilde;
P_corr = (eye(9) - K*Hk) * P_predict * (eye(9) - K*Hk)'
          + K*R*K';

% Kalman track positions and difference between Kalman
% and actual track position and actual target position
zout_K_track = H*x_corr;

track_diff = ztrue - zout_K_track;
track_error = [track_error, sqrt(track_diff'*
                                track_diff)];

% Update KF track trajectory array
K_track = [K_track, zout_K_track];

% Estimated accelerations
accel_out = [x_corr(3,:);
            x_corr(6,:);
            x_corr(9,:)];

% Update KF acceleration array
K_accel = [K_accel, accel_out];

end; % for ii = 2:nsamples

%*****
if kk == 1,
    % create first output

```

```

        zoutmean_true = zout_true_n;
        mean_K_track = K_track;
        merror_track = track_error;
        merror = error_true;

    else                                     % create output after 1st run

        zoutmean_true = zoutmean_true + zout_true_n;
        mean_K_track = mean_K_track + K_track;
        merror_track = merror_track + track_error;
        merror = merror + error_true;

    end; % if kk ==1, else
toc
end; % for kk = 1:nloops

%*****
% Compute Means
%*****
zoutmean_true = zoutmean_true/nloops;
mean_K_track = mean_K_track/nloops;
merror = merror/nloops; % mean error between
                        % measurement and true position

merror_track = merror_track/nloops; % mean error between
                                    % EKF estimated position
                                    % and true position

%*****
% Plot results
%*****
figure(1)
measurement = zoutmean_true/1000; % convert to km
Kalman_track = mean_K_track/1000; % convert to km
missile_track = missilevec(:,1:nsamples)/1000; % convert to km

plot3(missile_track(2,:),missile_track(5,:),missile_track(8,:),'g-',...
      Sensor_posit(1)/1000, Sensor_posit(2)/1000,...
      Sensor_posit(3)/1000,'rx');

axis([0,150,0,150,0,150]);
title('Ballistic Missile Base Trajectory - 120 seconds');
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
%print -deps c3pls1

figure(2)
plot3(missile_track(2,:),missile_track(5,:),missile_track(8,:),g-',...
      measurement(1,:), measurement(2,:), measurement(3,:),r-',...
      Sensor_posit(1)/1000,Sensor_posit(2)/1000, Sensor_posit(3)/1000,
      'rx');

axis([0,150,0,150,0,150]);

```

```

    title('Ballistic Missile Base Trajectory with Measurement Noise - 120
          seconds');
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    %print -deps c3pls2

figure(3)
    plot3(missile_track(2,1:nsamples), missile_track(5,1:nsamples), ...
          missile_track(8,1:nsamples), 'g-', ...
          Kalman_track(1,:), Kalman_track(2,:), Kalman_track(3,:), 'r-', ...
          Sensor_posit(1)/1000, Sensor_posit(2)/1000, Sensor_posit(3)/1000,
          'rx');

    axis([0,150,0,150,0,150]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    title('Ballistic Missile Base Trajectory and EKF Trajectory - 120
          seconds');
    %print -deps c3pls3

figure(4)
    start_pt = 1;
    stop_pt = 801;
    zoom_missile = [(start_pt + 1) : (stop_pt)];
    zoom_Kalman = [start_pt : stop_pt-1];
    plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
          missile_track(8, zoom_missile), 'g-', ...
          Kalman_track(1, zoom_Kalman), Kalman_track(2, zoom_Kalman),
          Kalman_track(3, zoom_Kalman), 'r-');

    axis([30,60,30,60,0,60]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    title(['ZOOM - EKF Trajectory Initial ', num2str((stop_pt -
          start_pt)/10), ' Seconds']);
    %print -deps c3pls4

figure(5)
    time = missilevec(1,:);
    diff_k_base = [Kalman_track(1,:) - missile_track(2,2:1200);
                  Kalman_track(2,:) - missile_track(5,2:1200);
                  Kalman_track(3,:) - missile_track(8,2:1200)];

    plot(time(2:nsamples), merror, 'g-', ...
          time(2:nsamples), 1000*sqrt(diff_k_base(1,:).^2 +
          diff_k_base(2,:).^2 + diff_k_base(3,:).^2), 'r-');

    xlabel('Time (seconds)'), ylabel('Error (meters)'), grid;
    title('EKF Distance Error vs. Time');
    legend('Mean Distance Error', 'EKF Distance Error');
    %print -deps c3pls5

```


APPENDIX C. SOURCE CODE FOR ALPHA-BETA-GAMMA TRACKING ALGORITHM

The following is the MATLAB[®] program used in the tracking of the ballistic missile base trajectory .

```
%*****
% abg.m
% LT Tony San Jose
% Thesis Advisor: R.G Hutchins
% 03FEB98
%
% This program uses an Alpha-Beta-Gamma tracker to filter the sensor
% measurement noise from the ballistic missile base trajectory
% developed using SIMULINK. Random noise is added to the measurement
% process. Actual missile track is generated in FlatEarthMissile
% SIMULINK model.
%
% delta = 0.1 sec
% nloops = 100
% alpha = 0.6
%*****

% Load base trajectory simulation workspace
clear all
load dat1;      % base trajectory developed in SIMULINK model
missilevec = missilevec';

% Define the number of simulation loops
nloops = 100;

% Define the sampling interval
delta = .1;

% Define the number of samples
[num_rows,num_cols] = size(missilevec);
nsamples = 1200;

% Initialize sensor data
Sensor_posit = [ 100 * 1000;      % sensor is 100 km in x
                 100 * 1000;      % sensor is 100 km in y
                 0 * 1000];      % sensor is 100 km in z

sigma_r = 10;          % 10 meters std dev in range
sigma_b = 1*pi/180;    % 1 degree std dev in azimuth
sigma_e = 1*pi/180;    % 1 degree std dev in elevation

% Define F matrix (TRANSITION MATRIX) for discrete time
% target motion,  $x(k+1) = F(k)*x(k) + G$ 
```

```

f_sub = [1, delta, (delta^2)/2;
         0, 1, delta;
         0, 0, 1];

F = [ f_sub, zeros(3), zeros(3);
      zeros(3), f_sub, zeros(3);
      zeros(3), zeros(3), f_sub ];

% Define G matrix
G = -g * [0;
          0;
          0;
          0;
          0;
          0;
          (delta^2)/2;
          delta;
          0];

% Define the H matrix (MEASUREMENT MATRIX), assuming that the
% x, y, and z missile positions are observed directly; z(k) = H(k)*x(k)

H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
      0, 0, 0, 1, 0, 0, 0, 0, 0;
      0, 0, 0, 0, 0, 0, 1, 0, 0];

% Define alpha, beta, gamma tracker parameters

alpha = 0.6;
beta = 2*(2-alpha) - 4*sqrt(1-alpha);
gamma = (beta^2)/(2*alpha);
nu = 1;

K_abg = [alpha, 0, 0;
         beta/(nu*delta), 0, 0;
         gamma/((nu*delta)^2), 0, 0;
         0, alpha, 0;
         0, beta/(nu*delta), 0;
         0, gamma/((nu*delta)^2), 0;
         0, 0, alpha;
         0, 0, beta/(nu*delta);
         0, 0, gamma/((nu*delta)^2)];

% Define initialization parameters

d_sub = [ 1, 0, 0, 0, 0, 0, 0;
          3/(2*delta), 0, 0, -2/delta, 0, 0, 1/(2*delta);
          1/(delta^2), 0, 0, -2/(delta^2), 0, 0, 1/delta^2];

D = [d_sub, zeros(3,2);
      zeros(3,1), d_sub, zeros(3,1);
      zeros(3,2), d_sub];

```



```

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
for kk = 1: nloops
tic
kk
    % define empty output matrices

    % measurement positions (cartesian) w/error
    zout_true_n = [];

    % distance error between measurement and true position
    error_true = [];

    % Kalman estimated trajectory
    ABG_track = [];

    % error between Kalman track and actual track
    track_error = [];

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
    for ii = 1:nsamples

        % Process the measurement from Sensor

        % True missile position
        ztrue = [missilevec(2,ii);
                 missilevec(5,ii);
                 missilevec(8,ii)];

%*****
% convert current position to polar coordinates and add
% sensor noise to the position, generating a noisy measurement
% from the sensor.
%*****

        % position relative to the sensor
        zrel = ztrue - Sensor_posit;

        r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2);
        % range from sensor
        □
        b = atan2(zrel(2), zrel(1));
        % bearing from sensor
        □

```

```

    r_prime = sqrt(zrel(1)^2 + zrel(2)^2);
    % range in x/y plane
    □
    e = atan2(zrel(3), r_prime);
    % elevation from sensor

% add noise to the measurement
    r_n = r + sigma_r * randn;
    b_n = b + sigma_b * randn;
    e_n = e + sigma_e * randn;

% measurement in polar + noise
    z_polar_n = [r_n;
                  b_n;
                  e_n];

% measurement in cartesian coordinates + noise
    z_cart_true_n = [r_prime*cos(b_n);
                     r_prime*sin(b_n);
                     r_n*sin(e_n) ] + Sensor_posit;

    z_cart_rel_n = [r_prime*cos(b_n);
                    r_prime*sin(b_n);
                    r_n*sin(e_n) ];

% compute measurement error in cartesian coordinates
    zdiff = ztrue - z_cart_true_n;
    disterror = sqrt(zdiff'*zdiff);

% Update the measurement array
    % true cartesian measurement + error
    zout_true_n = [zout_true_n, z_cart_true_n];

    % measurement error (between true measurement & true
    measurement w/noise)
    error_true = [error_true, disterror];

if ii > 2 % For intialization from the first 3 measurements

    %*****
    % Prediction
    %*****

% Initialization using the first 3 measurements
    if ii == 3
        x_corr = D * [zout_true_n(:,3);
                       zout_true_n(:,2);
                       zout_true_n(:,1)];
        end; %if ii==3

% ABG Filter prediction equations
    x_predict = F * x_corr + G;

```

```

%*****
% Correction
%*****

% Convert to relative position to compute RBE
coordinates
    x_1 = x_predict(1) - Sensor_posit(1);
    x_4 = x_predict(4) - Sensor_posit(2);
    x_7 = x_predict(7) - Sensor_posit(3);

% Convert prediction to Range, Bearing, Elevation
coordinates
    r_hat = sqrt(x_1^2 + x_4^2 + x_7^2);
    b_hat = atan2(x_4, x_1);
    e_hat = atan2(x_7, sqrt(x_1^2 + x_4^2));

% Determine expected measurement
    z_cart_exp_rel = [r_hat*cos(b_hat)*cos(e_hat);
                     r_hat*cos(e_hat)*sin(b_hat);
                     r_hat*sin(e_hat)];

    z_cart_exp_true = z_cart_exp_rel + Sensor_posit;

% Observed minus expected measurements
    z_tilde_c = z_cart_rel_n - z_cart_exp_rel;

% Correction equations
    x_corr = x_predict + K_abg * z_tilde_c;

% Alpha-Beta-Gamma track positions and difference
between ABG and
% actual track position and actual target position
    zout_ABG_track = H * x_corr;

    track_diff = ztrue - zout_ABG_track;
    track_error = [track_error,
                   sqrt(track_diff'*track_diff)];

% Update ABG track trajectory array
    ABG_track = [ABG_track, zout_ABG_track];

end; % if ii>2

end; % for ii = 1:nsamples

%*****

if kk == 1, % create first output

    zoutmean_true = zout_true_n;
    mean_ABG_track = ABG_track;

```

```

        merror_track = track_error;
        merror = error_true;

    else                                % create output after 1st run

        zoutmean_true = zoutmean_true + zout_true_n;
        mean_ABG_track = mean_ABG_track + ABG_track;
        merror_track = merror_track + track_error;
        merror = merror + error_true;

    end; % if kk ==1, else
toc
end; % for kk = 1:nloops

%*****
% Compute Means
%*****
    zoutmean_true = zoutmean_true/nloops;
    mean_ABG_track = mean_ABG_track/nloops;
    merror = merror/nloops;           % mean error between
                                     % measurement and true position

    merror_track = merror_track/nloops; % mean error between
                                     % EKF estimated position
                                     % and true position

%*****
% Plot results
%*****
figure(1)
    measurement = zoutmean_true/1000; % convert to km
    ABG = mean_ABG_track/1000; % convert to km
    missile_track = missilevec(:,1:nsamples)/1000; % convert to km

    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-
    ', ...
        Sensor_posit(1)/1000, Sensor_posit(2)/1000,
        Sensor_posit(3)/1000, 'rx');

    axis([0,150,0,150,0,150]);
    title('Ballistic Missile Base Trajectory - 120 seconds');
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    print -deps c4flc

figure(2)
plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-', ...
    measurement(1,:), measurement(2,:), measurement(3,:), 'r-', ...
    Sensor_posit(1)/1000, Sensor_posit(2)/1000, Sensor_posit(3)/1000, 'rx');

    axis([0,150,0,150,0,150]);
    title('Ballistic Missile Base Trajectory with Measurement Noise - 120
seconds');
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;

```

```

print -deps c4f2c

figure(3)
plot3(missile_track(2,:),missile_track(5,:),missile_track(8,:), 'g-',...
ABG(1,1:nsamples-2 ), ABG(2,1:nsamples-2), ABG(3,1:nsamples-2), 'r-',...
Sensor_posit(1)/1000,Sensor_posit(2)/1000,Sensor_posit(3)/1000,'rx');

axis([0,150,0,150,0,150]);
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
title('Ballistic Missile Base Trajectory and ABG Trajectory - 120
seconds');
print -deps c4f3c

figure(4)
start_pt = 1;
stop_pt = 401;
zoom_missile = [(start_pt + 1 ) : (stop_pt )];
zoom_Kalman = [start_pt : stop_pt-1];
plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
missile_track(8, zoom_missile), 'g-',...
ABG(1, zoom_Kalman), ABG(2, zoom_Kalman), ABG(3, zoom_Kalman), 'r-');

axis([30,60,30,60,0,60]);
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
title(['ZOOM - ABG Trajectory Initial ', num2str((stop_pt -
start_pt)/10), ' Seconds']);
print -deps c4f4c

figure(5)
start_pt = 1;
stop_pt = 601;
zoom_missile = [(start_pt + 1 ) : (stop_pt )];
zoom_Kalman = [start_pt : stop_pt-1];
plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
missile_track(8, zoom_missile), 'g-',...
ABG(1, zoom_Kalman), ABG(2, zoom_Kalman), ABG(3, zoom_Kalman), 'r-');

axis([30,60,30,60,0,60]);
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
title(['ZOOM - ABG Trajectory Initial ', num2str((stop_pt -
start_pt)/10), ' Seconds']);
print -deps c4f5c

figure(6)
start_pt = 1;
stop_pt = 801;
zoom_missile = [(start_pt + 1 ) : (stop_pt )];
zoom_Kalman = [start_pt : stop_pt-1];
plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
missile_track(8, zoom_missile), 'g-',...
ABG(1, zoom_Kalman), ABG(2, zoom_Kalman), ABG(3, zoom_Kalman), 'r-');

axis([30,60,30,60,0,60]);
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;

```

```

title(['ZOOM - ABG Trajectory Initial ',num2str((stop_pt -
start_pt)/10),' Seconds']);
print -deps c4f6c

figure(7)
time = missilevec(1,:);
diff_ABG_base = [ABG(1,:) - missile_track(2,3:nsamples);
                  ABG(2,:) - missile_track(5,3:nsamples);
                  ABG(3,:) - missile_track(8,3:nsamples)];

plot(time(1:nsamples), merror, 'g-',...
      time(3:nsamples), 1000*sqrt(diff_ABG_base(1,:).^2 +
      diff_ABG_base(2,:).^2 + diff_ABG_base(3,:).^2),'r-');

xlabel('Time (seconds)'), ylabel('Error (meters)'), grid;
title('ABG Distance Error vs. Time');
%axis([2 ,120, 0, 3000]);
legend('Mean Distance Error','ABG Distance Error');
print -deps c4f7c

figure(8)
plot(time(1:nsamples),merror,'g-',time(3:nsamples),merror_track,'r-');
xlabel('Time (seconds)'),ylabel('Mean Error (meters)'),grid,title('Mean
Distance Error in Measurements vs Time');% ('', num2str(nloops),' runs,
'',num2str(nsamples),' data points)'] ),grid;
print -deps c4f8c

```

APPENDIX D. SOURCE CODE FOR INTERACTING MULTIPLE MODEL TRACKING ALGORITHM

The following is the MATLAB[®] program used in the tracking of the ballistic missile base trajectory .

```
%*****
% imm.m
% LT Tony San Jose
% Thesis Advisor: R.G Hutchins
% 03FEB98
%
% This program generates a Kalman filter missile track using IMM with
% 2 models: an accelerating model and a ballistic model.
%
% The filter is initialized is from the missile launch position.
% Random noise is added in the sensor measurement process.
% Actual missile track is generated in FlatEarthMissile SIMULINK model.
%*****

% Load simulation workspace
clear all
load dat1;
missilevec = missilevec';

% Define the number of simulation loops
nloops = 100;

% Define the sampling interval
delta =.1;

% Define the number of samples
[num_rows,num_cols] = size(missilevec);
nsamples = 1200;

% Define q^2
q_sqr = 10;

% Initialize sensor data
Sensor_posit =[ 100 * 1000;    % sensor is 100 km in x
                100 * 1000;    % sensor is 100 km in y
                0 * 1000];    % sensor is 0 km in z

sigma_r = 10;                % 10 meters std dev in range
sigma_b = 1*pi/180;          % 1 degree std dev in azimuth
sigma_e = 1*pi/180;          % 1 degree std dev in elevation
```

```

R = diag([sigma_r^2,          % covariance matrix for
          sigma_b^2,          % uncorrelated
          sigma_e^2]);        % range and bearing measurements

% Define the H matrix (MEASUREMENT MATRIX) for the accelerating
% model

H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0, 0];

%*****
% ACCELERATING MODEL
%*****
% Define G matrix
G_accel = -g * [0;
                0;
                0;
                0;
                0;
                0;
                (delta^2)/2;
                delta;
                0];

% Initialize Q, the covariance of the plant noise

Q_sub_a = [(delta^5)/20, (delta^4)/8, (delta^3)/6;
            (delta^4)/8, (delta^3)/3, (delta^2)/2;
            (delta^3)/6, (delta^2)/2, delta ];

Q_accel = q_sqr * [Q_sub_a, zeros(3), zeros(3);
                   zeros(3), Q_sub_a, zeros(3);
                   zeros(3), zeros(3), Q_sub_a ];

% Define F matrix (TRANSITION MATRIX) for discrete time
% accelerating model.

f_sub_a = [1, delta, (delta^2)/2;
           0, 1, delta;
           0, 0, 1];

F_accel = [f_sub_a, zeros(3), zeros(3);
           zeros(3), f_sub_a, zeros(3);
           zeros(3), zeros(3), f_sub_a ];

%*****
% BALLISTIC MODEL
%*****
% Define G matrix
G_ball = -g * [0;
               0;
               0];

```



```

        0;
        0;
        0;
        (delta^2)/2;
        delta;
        0];

% Detemine Q for the Ballistic model

Q_sub_b = [(delta^3)/3, (delta^2)/2, 0;
            (delta^2)/2, delta, 0;
            0, 0, 0];

Q_ball = q_sqr * [ Q_sub_b, zeros(3), zeros(3);
                   zeros(3), Q_sub_b, zeros(3);
                   zeros(3), zeros(3), Q_sub_b];

% Define F matrix (TRANSITION MATRIX) for discrete time
% ballistic model.
f_sub_b = [1, delta, 0;
           0, 1, 0;
           0, 0, 0];

F_ball = [f_sub_b, zeros(3), zeros(3);
          zeros(3), f_sub_b, zeros(3);
          zeros(3), zeros(3), f_sub_b];

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
for kk = 1: nloops

tic
kk

    % define empty output matrices

    % measurement positions (cartesian) w/error
    zout_true_n = [];

    % distance error between measurement and true position
    error_true = [];

    % Kalman estimated trajectory
    K_track = [];
    K_accel = [];

    % error between Kalman track and actual track
    track_error = [];

    %*****
    % This block is used for the initialization process. Initialize
    % from launch position.

```

```

%*****
x_corr_accel = missilevec(2:10,1);
P_corr_accel = eye(9) * 10^4;

x_corr_ball = missilevec(2:10,1);
P_corr_ball = eye(9) * 10^4;

% Initial likelihoods for states.
mu_init = [1;
           0];

mu = mu_init;
mu_1 = mu(1);
mu_2 = mu(2);

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
for ii = 2:nsamples

    % Process the measurement from Sensor

    % True missile position
    ztrue = [missilevec(2,ii);
             missilevec(5,ii);
             missilevec(8,ii)];

%*****
% convert current position to polar coordinates and add
% sensor noise to the position, generating a noisy measurement
% from the sensor.
%*****

% position relative to the sensor
zrel = ztrue - Sensor_posit;

r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2); % range
                                           from sensor

b = atan2(zrel(2), zrel(1));                % bearing
                                           from sensor

r_prime = sqrt(zrel(1)^2 + zrel(2)^2);      % range in
                                           x/y plane

e = atan2(zrel(3), r_prime);                % elevation
                                           from sensor

% add noise to the measurement

```

```

        r_n = r + sigma_r * randn;
        b_n = b + sigma_b * randn;
        e_n = e + sigma_e * randn;

% measurement in polar + noise
        z_polar_n = [r_n;
                     b_n;
                     e_n];

% measurement in cartesian coordinates + noise
        z_cart_rel_n = [r_prime*cos(b_n);
                       r_prime*sin(b_n);
                       r_n*sin(e_n)  ];

        z_cart_true_n = z_cart_rel_n + Sensor_posit;

% compute measurement error in cartesian coordinates
        zdifff = ztrue - z_cart_true_n;
        disterror = sqrt(zdifff'*zdifff);

% Update the measurement array
        % true cartesian measurement + error
        zout_true_n = [zout_true_n, z_cart_true_n];

        % measurement error (between true measurements)
        error_true = [error_true, disterror];

%*****
% Prediction
%*****
        % Probabilities of changing state, Markov chain
        transition

□         p1 = 1;

□         p2 = 0.5;

□         alt = 50e3;

□         h = z_cart_true_n(3);

□

        probb_accel = -p2*( 1/(1+exp(-.0005*(h-alt))) - (1+p1) );
        probb_ball = 1 - probb_accel;

        rho = [probb_accel,  probb_ball;
               0,           1      ];

% Accelerating Model
        cbar = rho' * mu;

        if cbar(1) < 10^(-8)                % prevents cbar_1 from
                                                blowing up

```

```

        cbar_1 = 10^(-8);

    else
        cbar_1 = cbar(1);
    end;

    cbar_2 = cbar(2);

    rho_11 = rho(1,1);
    rho_21 = rho(2,1);
    rho_12 = rho(1,2);
    rho_22 = rho(2,2);

    x_hat_01 = x_corr_accel * ((rho_11*mu_1)/cbar_1) + ...
               x_corr_ball * ((rho_21*mu_2)/cbar_1);

    xtilde_11 = x_corr_accel - x_hat_01;
    xtilde_21 = x_corr_ball - x_hat_01;

    mu_11 = rho_11 * mu_1 / cbar_1;
    mu_21 = rho_21 * mu_2 / cbar_1;

    P_hat_01 = mu_11 * (P_corr_accel +
xtilde_11*xtilde_11') + ..
               mu_21 * (P_corr_ball +
xtilde_21*xtilde_21');

    % Kalman Filter Prediction Equations for Accelerating
model
    x_predict_accel = F_accel * x_hat_01 + G_accel;
    P_predict_accel = F_accel * P_hat_01 * F_accel' +
Q_accel;

    % Ballistic Model
    x_hat_02 = x_corr_accel * ((rho_12*mu_1)/cbar_2) + ...
               x_corr_ball * ((rho_22*mu_2)/cbar_2);

    xtilde_12 = x_corr_accel - x_hat_02;
    xtilde_22 = x_corr_ball - x_hat_02;

    mu_12 = rho_12 * mu_1 / cbar_2;
    mu_22 = rho_22 * mu_2 / cbar_2;

    P_hat_02 = mu_12*(P_corr_accel + xtilde_12*xtilde_12')
+ ...
               mu_22*(P_corr_ball + xtilde_22*xtilde_22');

    % Kalman Filter Prediction Equations for Ballistic model
    x_predict_ball = F_ball * x_hat_02 + G_ball;
    P_predict_ball = F_ball * P_hat_02 * F_ball' + Q_ball;

    %*****
    % Correction

```

```

%*****
% Accelerating Model
    % Convert to relative position to compute polar
coordinates
    x_1 = x_predict_accel(1) - Sensor_posit(1);
    x_4 = x_predict_accel(4) - Sensor_posit(2);
    x_7 = x_predict_accel(7) - Sensor_posit(3);

    % Convert prediction to polar coordinates
    r_hat_a = sqrt(x_1^2 + x_4^2 + x_7^2);
    b_hat_a = atan2(x_4, x_1);
    e_hat_a = atan2(x_7, sqrt(x_1^2 + x_4^2));

    % Determine expected measurement
    z_polar_hat_a = [r_hat_a;
                     b_hat_a;
                     e_hat_a];

    % Observed minus expected measurements
    z_tilde_a = z_polar_n - z_polar_hat_a;

    % The gradient of H evaluated at the most recent estimate
    Hk_r2c1 = -x_4/(x_1^2 + x_4^2);
    Hk_r2c4 = x_1/(x_1^2 + x_4^2);
    Hk_r3c1 = (-x_1*x_7)/(sqrt(x_1^2 + x_4^2)*(x_1^2 + x_4^2 + x_7^2));
    Hk_r3c4 = (-x_4*x_7)/(sqrt(x_1^2 + x_4^2)*(x_1^2 + x_4^2 + x_7^2));
    Hk_r3c7 = (sqrt(x_1^2 + x_4^2))/(x_1^2 + x_4^2 + x_7^2);

    % Determine H matrix
    Hk_a = [x_1/r_hat_a, 0, 0, x_4/r_hat_a, 0, 0, x_7/r_hat_a, 0, 0;
            Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
            Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

    % Compute Kalman Gain
    K_accel = P_predict_accel * Hk_a' * inv(Hk_a * P_predict_accel * Hk_a' + R);

    % Kalman Filter Correction equations for Accelerating Model
    x_corr_accel = x_predict_accel + K_accel * z_tilde_a;
    P_corr_accel = (eye(9) - K_accel * Hk_a) * P_predict_accel;

% Ballistic Model
    % Convert to relative position to compute polar
coordinates
    x_1 = x_predict_ball(1) - Sensor_posit(1);
    x_3 = x_predict_ball(4) - Sensor_posit(2);
    x_5 = x_predict_ball(7) - Sensor_posit(3);

    % Convert prediction to polar coordinates
    r_hat_b = sqrt(x_1^2 + x_3^2 + x_5^2);
    b_hat_b = atan2(x_3, x_1);
    e_hat_b = atan2(x_5, sqrt(x_1^2 + x_3^2));

```

```

% Determine expected measurement
z_polar_hat_b = [r_hat_b;
                  b_hat_b;
                  e_hat_b];

% Observed minus expected measurements
z_tilde_b = z_polar_n - z_polar_hat_b;

% The gradient of H evaluated at the most recent estimate
Hk_r2c1 = -x_3/(x_1^2 + x_3^2);
Hk_r2c4 = x_1/(x_1^2 + x_3^2);
Hk_r3c1 = (-x_1*x_5)/( (sqrt(x_1^2 + x_3^2))*(x_1^2 + x_3^2 + x_5^2) );
Hk_r3c4 = (-x_3*x_5)/( (sqrt(x_1^2 + x_3^2))*(x_1^2 + x_3^2 + x_5^2) );
Hk_r3c7 = (sqrt(x_1^2 + x_3^2))/(x_1^2 + x_3^2 + x_5^2);

% Determine H matrix
Hk_b = [x_1/r_hat_b, 0, 0, x_3/r_hat_b, 0, 0, x_5/r_hat_b, 0, 0;
        Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
        Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

% Compute Kalman Gain
K_ball = P_predict_ball * Hk_b' * inv(Hk_b * P_predict_ball * Hk_b' + R);

% Kalman Filter Correction equations for the Ballistic Model
x_corr_ball = x_predict_ball + K_ball * z_tilde_b;
P_corr_ball = (eye(9) - K_ball * Hk_b) * P_predict_ball;

%*****
% Update mode probabilities
%*****
m = 3;

S_1 = Hk_a * P_predict_accel * Hk_a' + R;
lambda_1 = (exp( -(z_tilde_a)' * inv(S_1) * z_tilde_a / 2 ) ) / (sqrt((2*pi)^m * det(S_1)));

S_2 = Hk_b * P_predict_ball * Hk_b' + R;
lambda_2 = ( exp( -(z_tilde_b)' * inv(S_2) * z_tilde_b / 2 ) ) / ( sqrt( (2*pi)^m * det(S_2) ) );

c = lambda_1 * cbar_1 + lambda_2 * cbar_2;

mu_1 = lambda_1 * cbar_1 / c;
mu_2 = lambda_2 * cbar_2 / c;

mu = [mu_1;
      mu_2];

%*****
% Produce Combined Estimates
%*****
x_corr = mu_1 * x_corr_accel + mu_2 * x_corr_ball;
P_corr = mu_1 * (P_corr_accel + (x_corr_accel -

```

```

        x_corr)*(x_corr_accel-x_corr')+...
mu_2*(P_corr_ball +(x_corr_ball-
        x_corr)*(x_corr_ball- x_corr)');

%*****
% Kalman track positions and difference between Kalman
% and actual track position and actual target position
zout_K_track = H*x_corr;

track_diff = ztrue - zout_K_track;
track_error = [track_error,
               sqrt(track_diff'*track_diff)];

% Update KF track trajectory array
K_track = [K_track, zout_K_track];

end; % for ii = 2:20:nsamples

%*****

if kk == 1, % create first output

    zoutmean_true = zout_true_n;
    mean_K_track = K_track;
    merror_track = track_error;
    merror = error_true;

else % create output after 1st run

    zoutmean_true = zoutmean_true + zout_true_n;
    mean_K_track = mean_K_track + K_track;
    merror_track = merror_track + track_error;
    merror = merror + error_true;

end; % if kk ==1, else

toc

end; % for kk = 1:nloops

%*****
% Compute Means
%*****
zoutmean_true = zoutmean_true/nloops;
mean_K_track = mean_K_track/nloops;
merror = merror/nloops; % mean error between
                        % measurement and true position

merror_track = merror_track/nloops; % mean error between
                                    % EKF estimated position
                                    % and true position

```

```

%*****
% Plot results
%*****
figure(1)
    measurement = zoutmean_true/1000;           % convert to km
    Kalman_track = mean_K_track/1000;           % convert to km
    missile_track = missilevec(:,1:nsamples)/1000; % convert to km

    plot3(missile_track(2,:),missile_track(5,:), missile_track(8,:),...
Sensor_posit(1)/1000,Sensor_posit(2)/1000,Sensor_posit(3)/1000,'rx');

    axis([0,150,0,150,0,150]);
    title('Ballistic Missile Base Trajectory - 120 seconds');
    xlabel('x - km'), ylabel('y - km'), zlabel('z - km'),grid;
print -deps ch5f1a

figure(2)
    plot3(missile_track(2,:),missile_track(5,:), missile_track(8,:),...
    measurement(1,:),measurement(2,:),measurement(3,:),...

Sensor_posit(1)/1000,Sensor_posit(2)/1000,Sensor_posit(3)/1000,'rx');

    axis([0,150,0,150,0,150]);
    title('Ballistic Missile Base Trajectory with Measurement Noise - 120
seconds');
    xlabel('x - km'), ylabel('y - km'), zlabel('z - km'),grid;
print -deps ch5f2a

figure(3)
    plot3(missile_track(2,:),missile_track(5,:),missile_track(8:),'g-
',...
    Kalman_track(1,:),Kalman_track(2,:),Kalman_track(3:),'r-');

    axis([0,150,0,150,0,150]);
    xlabel('x - km'), ylabel('y - km'), zlabel('z - km'),grid;
    title('Ballistic Missile Base Trajectory and IMM Trajectory - 120
seconds');
print -deps ch5f3a

figure(4)
    start_pt = 1;
    stop_pt = 401;
    zoom_missile = [(start_pt +1 ) : (stop_pt )];
    zoom_Kalman = [start_pt : stop_pt-1];
    plot3(missile_track(2,zoom_missile),missile_track(5,zoom_missile),
missile_track(8,zoom_missile),'g-',...
    Kalman_track(1,zoom_Kalman),
Kalman_track(2,zoom_Kalman),Kalman_track(3,zoom_Kalman),'r-');
    axis([30,60,30,60,0,60]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;

```



```

    title(['ZOOM - IMM Trajectory Initial ', num2str((stop_pt -
start_pt)/10), ' Seconds'] );
print -deps ch5f4a

```

```

figure(5)
    start_pt = 1;
    stop_pt = 601;
    zoom_missile = [(start_pt +1 ) : (stop_pt )];
    zoom_Kalman = [start_pt : stop_pt-1];
    plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
missile_track(8, zoom_missile), 'g-', ...
        Kalman_track(1, zoom_Kalman),
Kalman_track(2, zoom_Kalman), Kalman_track(3, zoom_Kalman), 'r-');
    axis([30, 60, 30, 60, 0, 60]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    title(['ZOOM - IMM Trajectory Initial ', num2str((stop_pt -
start_pt)/10), ' Seconds'] );
print -deps ch5f5a

```

```

figure(6)
    start_pt = 1;
    stop_pt = 801;
    zoom_missile = [(start_pt +1 ) : (stop_pt )];
    zoom_Kalman = [start_pt : stop_pt-1];
    plot3(missile_track(2, zoom_missile), missile_track(5, zoom_missile),
missile_track(8, zoom_missile), 'g-', ...
        Kalman_track(1, zoom_Kalman),
Kalman_track(2, zoom_Kalman), Kalman_track(3, zoom_Kalman), 'r-');
    axis([30, 60, 30, 60, 0, 60]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    title(['ZOOM - IMM Trajectory Initial ', num2str((stop_pt -
start_pt)/10), ' Seconds'] );
print -deps ch5f6a

```

```

figure(7)
    time = missilevec(1,:);
    diff_IMM_base = [Kalman_track(1,:) - missile_track(2,2:nsamples);
                    Kalman_track(2,:) - missile_track(5,2:nsamples);
                    Kalman_track(3,:) - missile_track(8,2:nsamples)];

    plot(time(2:nsamples), merror, 'g-', ...
        time(2:nsamples), 1000*sqrt(diff_IMM_base(1,:).^2 +
diff_IMM_base(2,:).^2 + diff_IMM_base(3,:).^2), 'r-');

    xlabel('Time (seconds)'), ylabel('Error (meters)'), grid;
    title('IMM Distance Error vs. Time');
    legend('Mean Distance Error', 'IMM Distance Error');
    print -deps c5f7a

```

```

figure(8)
    plot(time(2:nsamples), merror, 'g-', time(2:nsamples), merror_track, 'r-');

```

```
    xlabel('Time (seconds)'),ylabel('Mean Error  
(meters)'),grid,title('Mean Distance Error in Measurements vs Time');%  
    ('', num2str(nloops),' runs, ',num2str(nsamples),' data points)'] ),grid;  
    print -deps c5f8a
```

```
save imm100
```

APPENDIX E. TBM PROFILES

A. TBM PROFILE NUMBER 1

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	36.0	0.000	0.000	33	60.6	7.023	3.195
1	36.3	0.006	0.000	34	62.4	7.469	3.491
2	36.6	0.026	0.000	35	64.2	7.928	3.803
3	36.9	0.058	0.000	36	66.0	8.402	4.132
4	37.2	0.103	0.000	37	68.4	8.890	4.479
5	37.5	0.163	0.001	38	70.8	9.393	4.844
6	37.8	0.235	0.004	39	73.2	9.911	5.229
7	38.1	0.322	0.010	40	75.6	10.444	5.633
8	38.4	0.423	0.020	41	78.0	10.992	6.057
9	38.7	0.537	0.036	42	81.2	11.556	6.502
10	39.0	0.666	0.058	43	84.4	12.136	6.969
11	39.5	0.809	0.087	44	87.6	12.732	7.459
12	40.0	0.965	0.124	45	90.8	13.345	7.973
13	40.5	1.136	0.171	46	94.0	13.975	8.511
14	41.0	1.321	0.226	47	96.0	14.622	9.075
15	41.5	1.520	0.292	48	98.0	15.288	9.665
16	42.0	1.733	0.367	49	100.0	15.972	10.282
17	42.5	1.962	0.453	50	102.0	16.675	10.928
18	43.0	2.204	0.550	51	104.0	17.397	11.604
19	43.5	2.460	0.658	52	104.6	18.140	12.309
20	44.0	2.731	0.777	53	105.2	18.904	13.045
21	45.0	3.015	0.908	54	105.8	19.690	13.813
22	46.0	3.312	1.050	55	106.4	20.499	14.613
23	47.0	3.623	1.205	56	107.0	21.332	15.446
24	48.0	3.948	1.372	57	106.4	22.190	16.314
25	49.0	4.286	1.551	58	105.8	23.075	17.217
26	50.6	4.637	1.744	59	105.2	23.986	18.155
27	52.2	5.001	1.950	60	104.6	24.925	19.131
28	53.8	5.378	2.170	61	104.0	25.894	20.145
29	55.4	5.769	2.404	62	98.0	26.894	21.199
30	57.0	6.174	2.652	63	80.0	27.925	22.293
31	58.8	6.591	2.916	62.5	20.0	28.450	22.850

B. TBM PROFILE NUMBER 2

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	136.26	0.0000	0.0000	33	136.26	7.4687	3.4908
1	136.26	0.0064	0.0000	34	136.26	7.9283	3.8028
2	136.26	0.0256	0.0001	35	136.26	8.4021	4.1320
3	136.26	0.0579	0.0002	36	136.26	8.8904	4.4790
4	136.26	0.1035	0.0001	37	136.26	9.3933	4.8443
5	136.26	0.1626	0.0009	38	136.26	9.9111	5.2287
6	136.26	0.2355	0.0039	39	136.26	10.4440	5.6326
7	136.26	0.3222	0.0100	40	136.26	10.9922	6.0569
8	136.26	0.4228	0.0203	41	136.14	11.5560	6.5022
9	136.26	0.5374	0.0358	42	136.00	12.1358	6.9694
10	136.26	0.6661	0.0576	43	135.86	12.7319	7.4594
11	136.26	0.8087	0.0868	44	135.72	13.3448	7.9729
12	136.26	0.9653	0.1243	45	135.58	13.9748	8.5110
13	136.26	1.1359	0.1707	46	135.44	14.6224	9.0746
14	136.26	1.3207	0.2264	47	135.30	15.2879	9.6647
15	136.26	1.5199	0.2919	48	135.16	15.9718	10.2823
16	136.26	1.7335	0.3675	49	135.02	16.6746	10.9285
17	136.26	1.9615	0.4535	50	134.88	17.3969	11.6039
18	136.26	2.2038	0.5503	51	134.74	18.1396	12.3093
19	136.26	2.4602	0.6581	52	134.60	18.9036	13.054
20	136.26	2.7305	0.7771	53	134.46	19.6897	13.8131
21	136.26	3.0146	0.9078	54	134.32	20.4989	14.6132
22	136.26	3.3123	1.0502	55	134.18	21.3321	15.4465
23	136.26	3.6234	1.2047	56	133.43	22.1903	16.3140
24	136.26	3.9479	1.3717	57	130.50	23.0745	17.2166
25	136.26	4.2856	1.5513	58	127.00	23.9859	18.1553
26	136.26	4.6366	1.7439	59	121.00	24.9255	19.1312
27	136.26	5.0008	1.9499	60	111.00	25.8944	20.1453
28	136.26	5.3784	2.1697	61	86.00	26.8938	21.1987
29	136.26	5.7692	2.4037	62	65.00	27.9250	22.2926
30	136.26	6.1736	2.6524	63	20.00	28.9836	23.4225
31	136.26	6.5915	2.9161	64	0.00	30.0367	24.5560
32	136.26	7.0231	3.1954				

C. TBM PROFILE NUMBER 3

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	36.40	0.8230	0.0025	35	67.86	8.8360	4.5970
1	36.40	0.8291	0.0025	36	70.20	9.3060	4.9690
2	36.66	0.8478	0.0026	37	72.28	9.7900	5.3600
3	36.66	0.8791	0.0027	38	74.62	10.2900	5.7700
4	36.66	0.9231	0.0049	39	77.48	10.8000	6.2020
5	36.66	0.9796	0.0124	40	80.08	11.3300	6.6540
6	36.66	1.0490	0.0254	41	82.94	11.8800	7.1280
7	36.92	1.1310	0.0437	42	85.80	12.4400	7.6250
8	37.18	1.2260	0.0675	43	88.66	13.0100	8.1440
9	37.44	1.3350	0.0974	44	91.78	13.6100	8.6880
10	37.44	1.4570	0.1338	45	94.64	14.2200	9.2560
11	37.70	1.5920	0.1773	46	96.72	14.8500	9.8500
12	37.96	1.7400	0.2286	47	98.80	15.5000	10.4700
13	38.22	1.9020	0.2881	48	100.88	16.600	11.1200
14	38.74	2.0780	0.3564	49	102.18	16.8500	11.7900
15	39.52	2.2670	0.4339	50	103.48	17.5600	12.5000
16	40.30	2.4690	0.5211	51	104.52	18.2800	13.2300
17	41.34	2.6850	0.6186	52	105.56	19.0300	14.0000
18	42.38	2.9140	0.7268	53	106.60	19.8000	14.7900
19	43.42	3.1570	0.8462	54	107.38	20.5900	15.6200
20	44.46	3.4140	0.9771	55	108.42	21.4100	16.4800
21	45.50	3.6840	1.1200	56	109.20	22.2500	17.3800
22	46.80	3.9660	1.2750	57	109.72	23.1200	18.3100
23	48.10	4.2620	1.4430	58	109.98	224.0200	19.2700
24	49.40	4.5710	1.6240	59	98.28	24.9400	20.2700
25	50.96	4.8930	1.8180	60	86.32	25.8900	21.3100
26	52.26	5.2280	2.0270	61	52.26	26.8700	22.3800
27	53.82	5.5760	2.2490	62	14.12	27.8900	23.5000
28	55.38	5.9370	2.4860	63	8.11	28.9300	24.6500
29	56.94	6.3110	2.7390	64	6.08	30.0100	25.8500
30	58.76	6.6980	3.0060	65	5.93	31.1200	27.0900
31	60.32	7.0980	3.2900	66	5.80	32.2700	28.3700
32	62.14	7.5120	3.5910	67	5.80	33.4600	29.6900
33	63.96	7.9400	3.9080	68	5.80	34.6800	31.0700
34	65.78	8.3810	4.2430	69	5.80	35.9500	32.4900

D. TBM PROFILE NUMBER 4

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	36.66	0.0000	0.0000	35	50.96	4.4014	1.0029
1	36.66	0.0030	0.0000	36	52.26	4.6796	1.1067
2	36.66	0.0119	0.0000	37	53.56	4.9373	1.2177
3	36.66	0.0270	0.0001	38	54.86	5.2641	1.3360
4	36.66	0.0483	0.0003	39	56.16	5.5698	1.4619
5	36.66	0.0760	0.0008	40	57.72	5.8846	1.5956
6	36.66	0.1101	0.0018	41	59.28	6.2082	1.7373
7	36.66	0.1508	0.0032	42	60.58	6.5409	1.8872
8	36.66	0.1981	0.0052	43	62.14	6.08826	2.0457
9	36.66	0.2523	0.0080	44	63.70	7.2335	2.2130
10	36.66	0.3133	0.0118	45	65.26	7.5939	2.3895
11	36.66	0.3814	0.0166	46	66.82	7.9640	2.5756
12	36.92	0.4567	0.0226	47	68.64	8.3440	2.7716
13	36.92	0.5391	0.0302	48	70.46	8.7341	2.9779
14	36.92	0.6289	0.0393	49	72.28	9.1344	3.1949
15	36.92	0.7262	0.0502	50	74.36	9.5452	3.4228
16	37.18	0.8310	0.0632	51	76.44	9.9667	3.6621
17	37.18	0.9435	0.0784	52	78.78	10.3990	3.9121
18	37.44	1.0638	0.0961	53	81.12	10.8430	4.1764
19	37.70	1.1919	0.1164	54	83.72	11.2980	4.4521
20	37.96	1.3281	0.1396	55	86.32	11.7640	4.7409
21	38.22	1.4723	0.1659	56	88.92	12.2430	5.0430
22	38.74	1.6247	0.1956	57	91.52	12.7330	5.3589
23	39.26	1.7854	0.2289	58	94.12	13.2360	5.6891
24	39.78	1.9545	0.2660	59	96.98	13.7520	6.0339
25	40.56	2.1322	0.3073	60	99.84	14.2800	6.03938
26	41.34	2.3185	0.3529	61	101.92	14.8220	6.7693
27	42.38	2.5135	0.4031	62	103.74	15.3760	7.1607
28	43.16	2.7174	0.4582	63	105.30	15.9450	7.5686
29	44.20	2.9302	0.5184	64	106.60	16.5270	7.9933
30	45.24	3.1522	0.5840	65	107.90	17.1240	8.4354
31	46.28	3.3833	0.6553	66	108.94	17.7350	8.8954
32	47.32	3.6237	0.7326	67	109.72	18.3600	9.3738
33	48.36	3.8734	0.8161	68	110.76	19.0010	9.8710
34	49.66	4.1326	0.9061	69	111.54	19.6570	10.3880

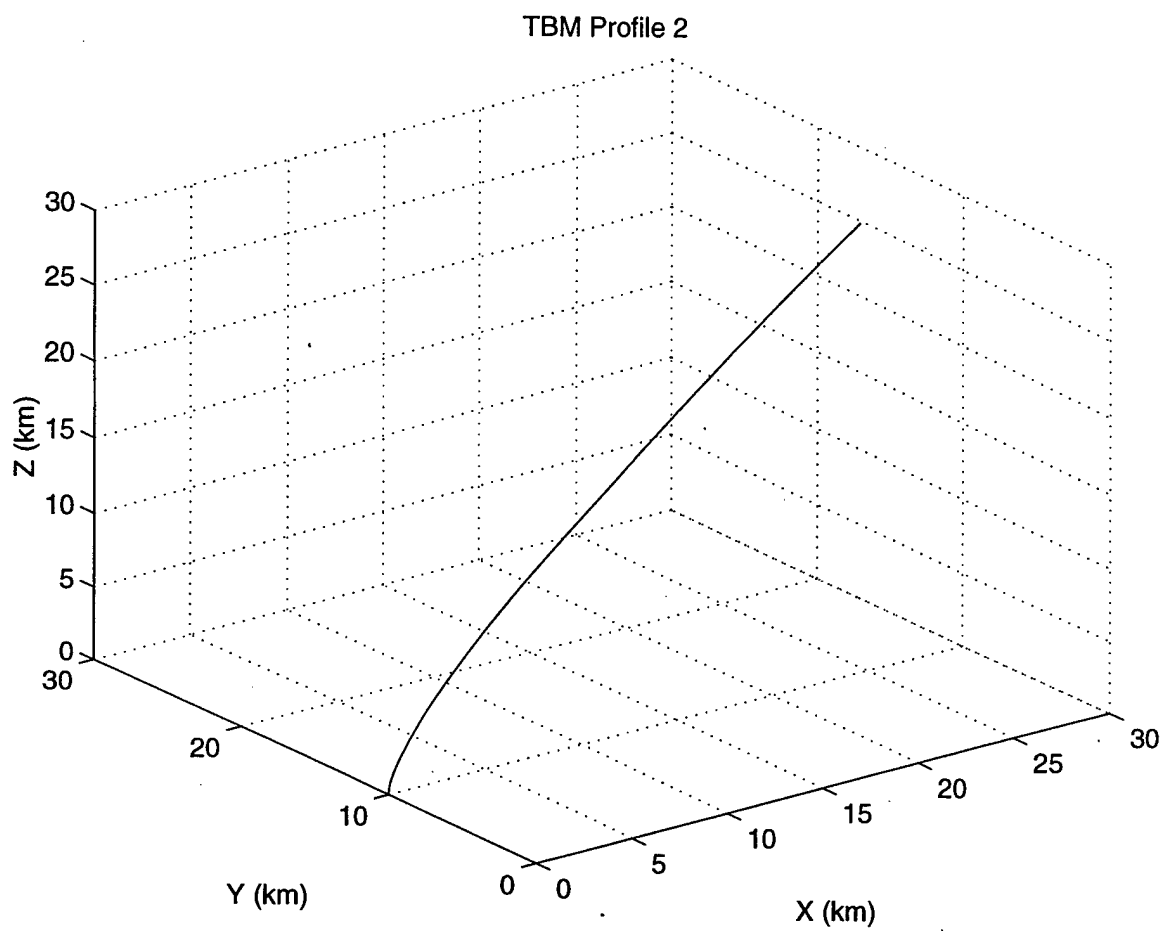
Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
70	112.32	20.3290	10.9240	81	7.59	28.8550	18.2970
71	113.10	21.0170	11.4810	82	6.45	29.7420	19.1160
72	114.14	21.7220	12.0590	83	6.24	30.6500	19.9620
73	114.92	22.4430	12.6590	84	6.11	31.5780	20.8360
74	115.44	23.1810	13.2800	85	6.08	32.5280	21.7390
75	115.96	23.9370	13.9240	86	6.08	33.5000	22.6720
76	112.06	24.7100	14.5920	87	6.08	34.4930	23.6340
77	100.62	25.5010	15.2830	88	6.08	35.5100	24.6270
78	81.90	26.3110	15.9980	89	6.08	36.5510	25.6510
79	39.52	27.1390	16.7390	90	6.08	37.6160	26.7070
80	10.40	27.9870	17.5050	91	6.08	38.7060	27.7950

E. TBM PROFILE NUMBER 5

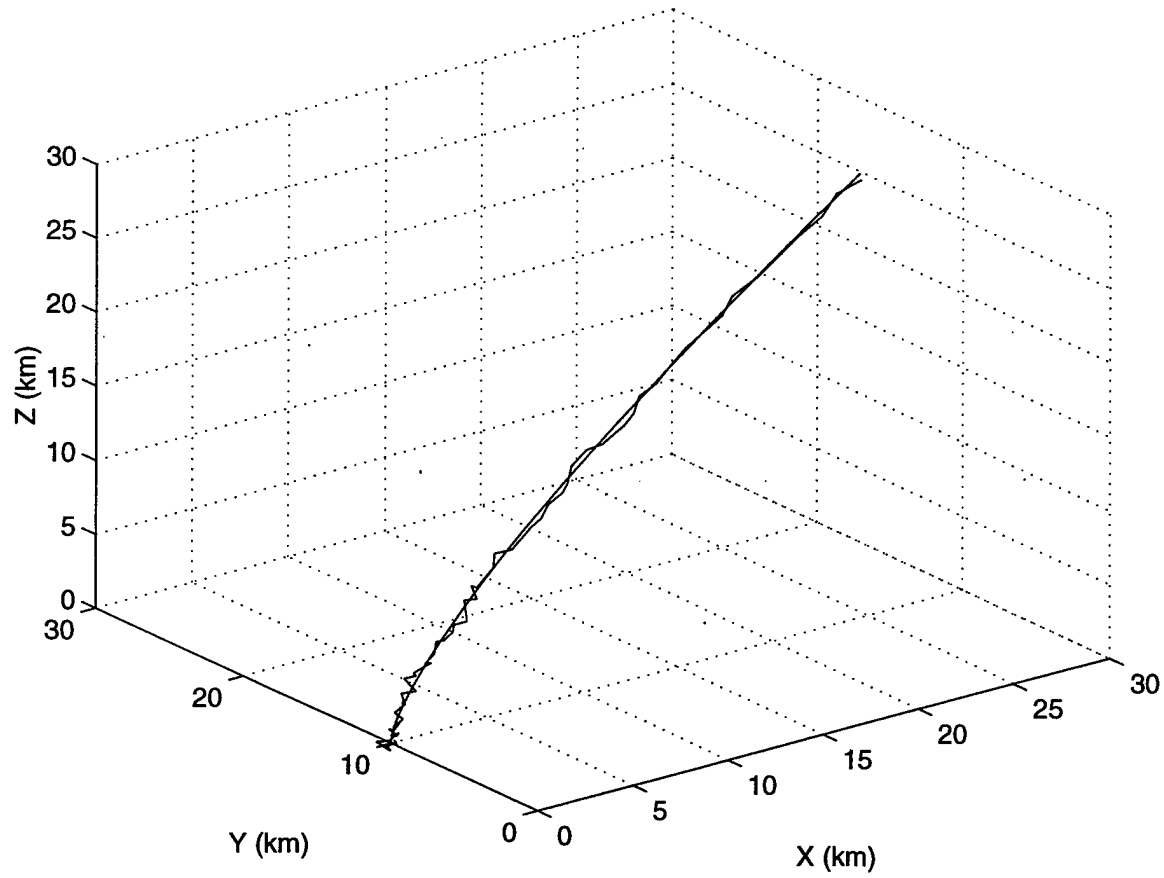
Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
0	136.26	0.0000	0.0000	35	136.26	7.2367	3.5962
1	136.26	0.0054	0.0000	36	136.26	7.6546	3.9031
2	136.26	0.0217	0.0001	37	136.26	8.0846	4.2263
3	136.26	0.0491	0.0002	38	136.26	8.8257	4.5663
4	136.26	0.0879	0.0003	39	136.26	8.9791	4.9237
5	136.26	0.1381	0.0012	40	136.26	9.4446	5.2990
6	136.26	0.1997	0.0053	41	136.14	9.9225	5.6926
7	136.26	0.2731	0.0130	42	136.00	10.4127	6.1051
8	136.26	0.3582	0.0246	43	135.86	10.9156	6.5370
9	136.26	0.4551	0.0409	44	135.72	11.4315	6.9889
10	136.26	0.5640	0.0623	45	135.58	11.9604	7.4615
11	136.26	0.6851	0.0894	46	135.44	12.0529	7.9554
12	136.26	0.8183	0.1225	47	135.30	13.0591	8.4713
13	136.26	0.9638	0.1624	48	135.16	13.6294	9.0098
14	136.26	1.1216	0.2094	49	135.02	14.2143	9.5718
15	136.26	1.2920	0.2641	50	134.88	14.8140	10.1580
16	136.26	1.4749	0.3269	51	134.74	15.4290	10.7689
17	136.26	1.6705	0.3984	52	134.60	16.0600	11.4053
18	136.26	1.8787	0.4790	53	134.46	16.7076	12.0676
19	136.26	2.0996	0.5691	54	134.32	17.3724	12.7565
20	136.26	2.3330	0.6692	55	134.18	18.0552	13.4725
21	136.26	2.5788	0.7796	56	133.43	18.7565	14.2162
22	136.26	2.8366	0.9006	57	130.50	19.4772	14.9884
23	136.26	3.1065	1.0325	58	127.00	20.2178	15.7896
24	136.26	3.3881	1.1758	59	121.00	20.9793	16.6207
25	136.26	3.6814	1.3308	60	111.00	21.7624	17.4823
26	136.26	3.9862	1.4976	61	100.00	22.5678	18.3753
27	136.26	4.3024	1.6768	62	85.00	23.3965	19.3003
28	136.26	4.6298	1.8685	63	62.00	24.2493	20.2583
29	136.26	4.9685	2.0733	64	42.00	25.1271	21.2500
30	136.26	5.3183	2.2914	65	30.00	26.0308	22.2765
31	136.26	5.6794	2.5232	66	22.00	26.9614	23.3385
32	136.26	6.0517	2.7692	67	16.00	27.9199	24.4370
33	136.26	6.4353	3.0297	68	14.00	28.9074	25.5730
34	136.26	6.8303	3.3053	69	12.50	29.9247	26.7476

Time (sec)	Intensity	Altitude (km)	Range (km)	Time (sec)	Intensity	Altitude (km)	Range (km)
70	11.00	30.9732	27.9617	76	7.10	37.9850	36.1414
71	10.00	32.0539	29.2164	77	6.80	39.2863	37.6656
72	9.30	33.1681	30.5130	78	6.40	40.6296	39.2395
73	8.60	34.3169	31.8525	79	6.10	42.0164	40.8921
74	8.10	35.5018	33.2362	80	5.80	43.4485	42.5725
75	7.60	36.7240	34.6653	81	0.00	44.9228	44.3028

F. TBM PROFILE 2 ANALYSIS

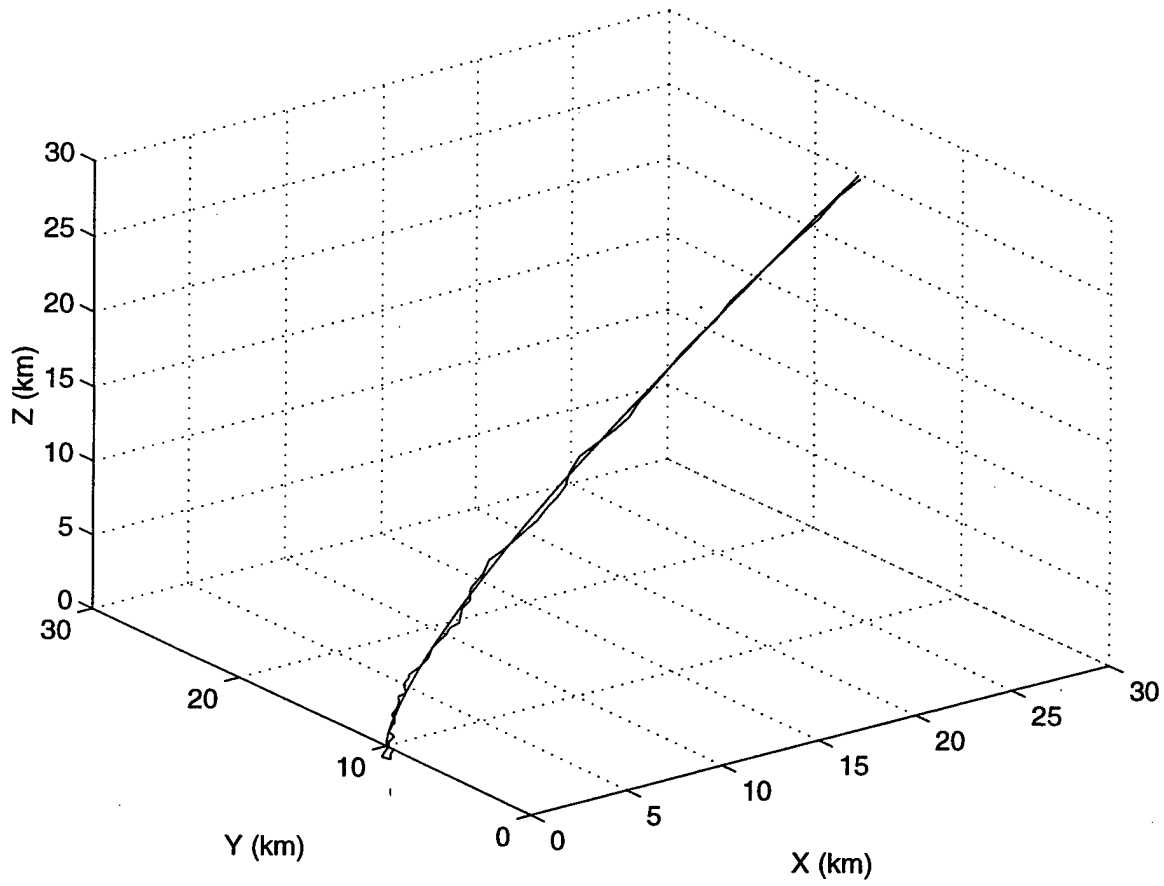


TBM Profile 2 w/ Measurement Noise

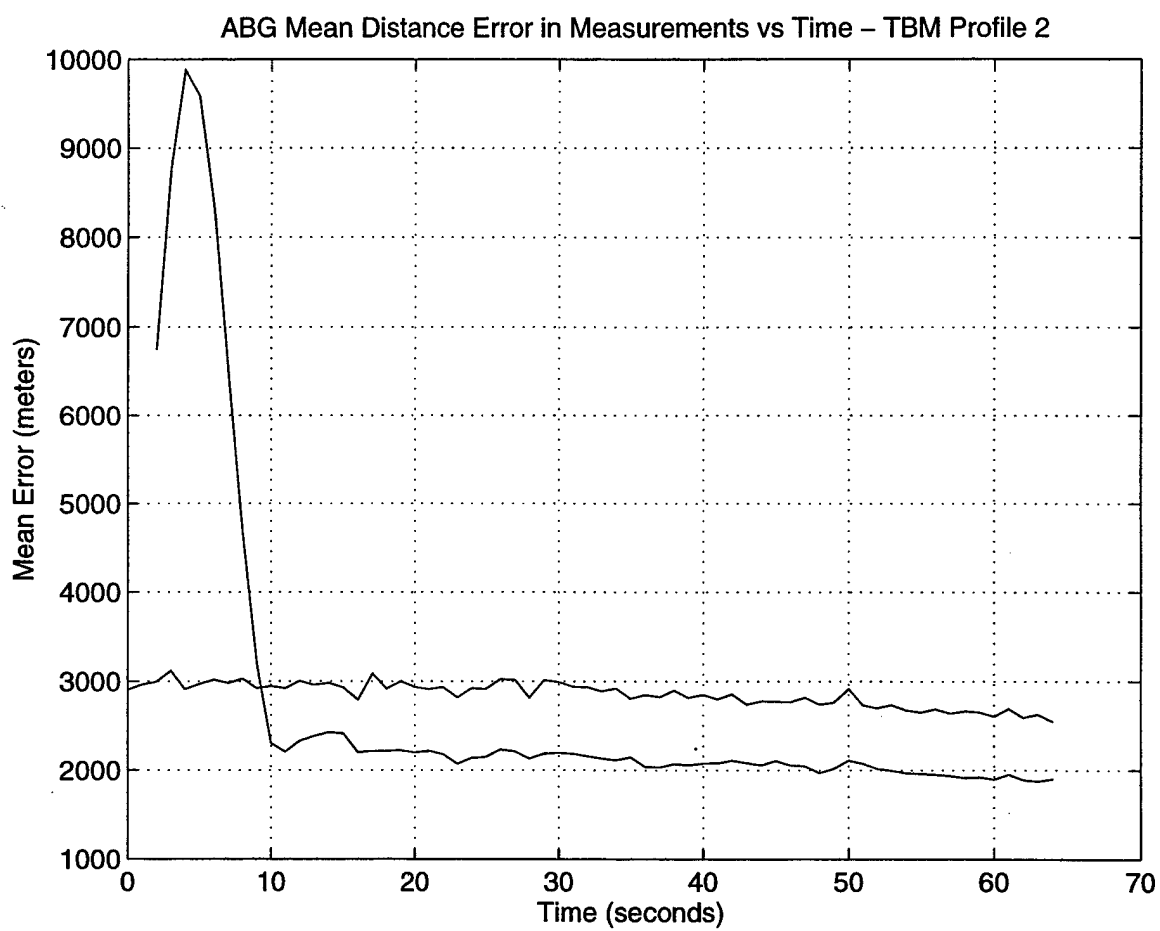


TBM Trajectory (Profile 2) with Measurement Noise, 100 Runs.

TBM Profile 2 and ABG Trajectory

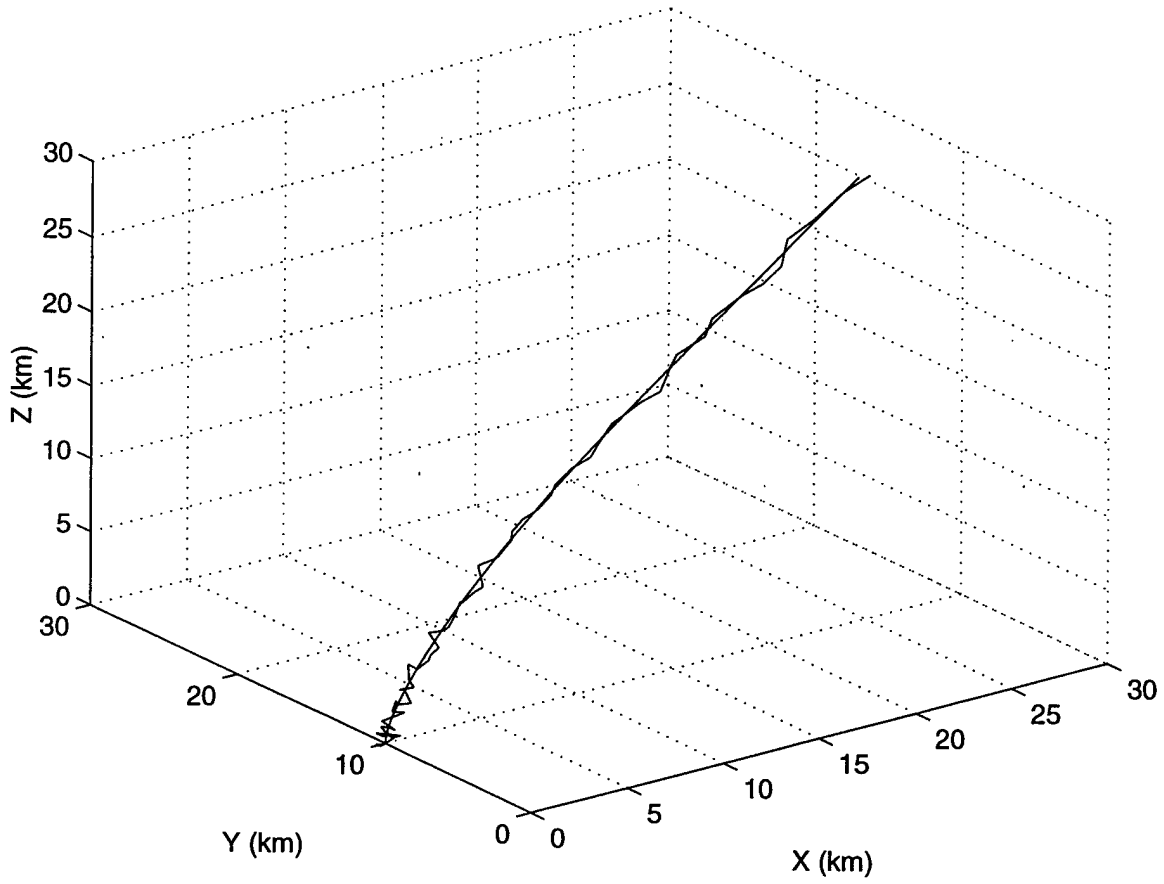


TBM Trajectory (Profile 2) and α - β - γ Trajectory, $\alpha=0.6$, 100 Runs.



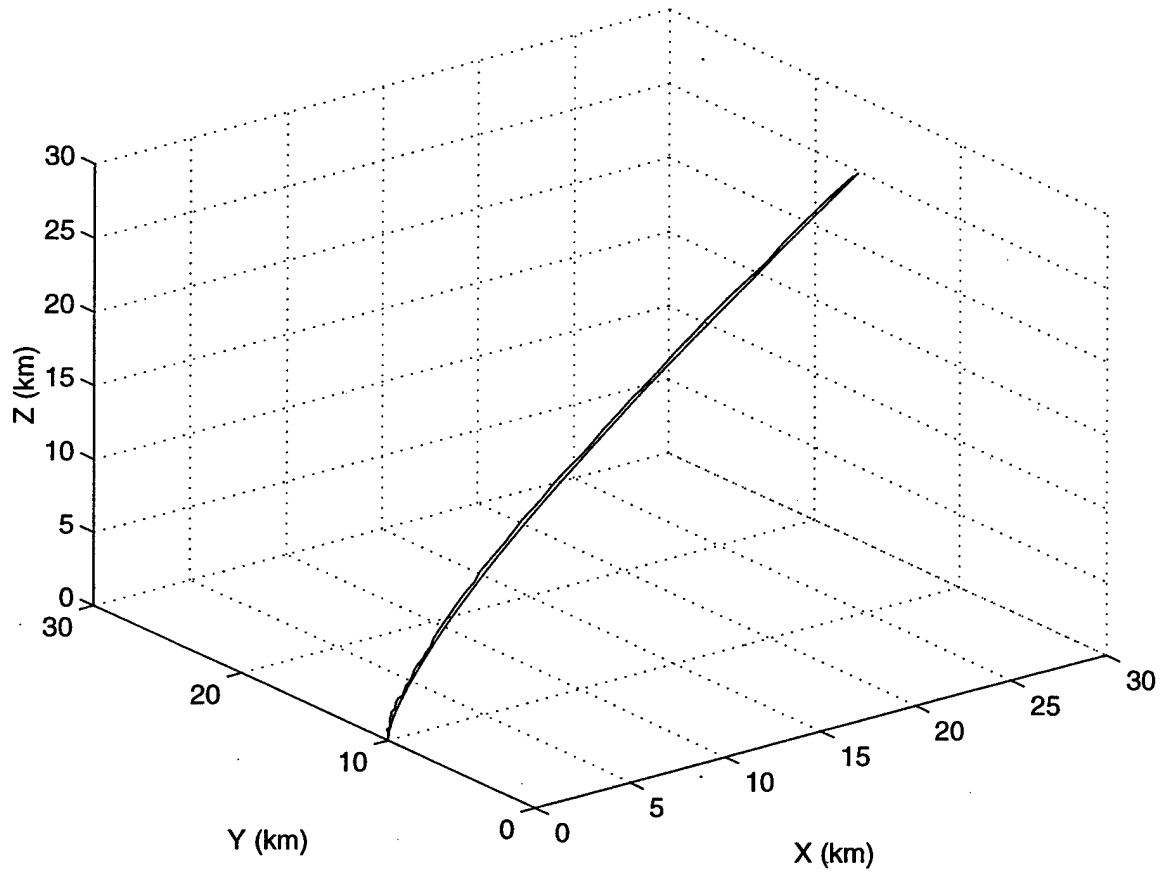
α - β - γ Tracker (Profile 2) Mean Distance Error, $\alpha=0.6$, 500 Runs.

TBM Profile 2 w/ Measurement Noise

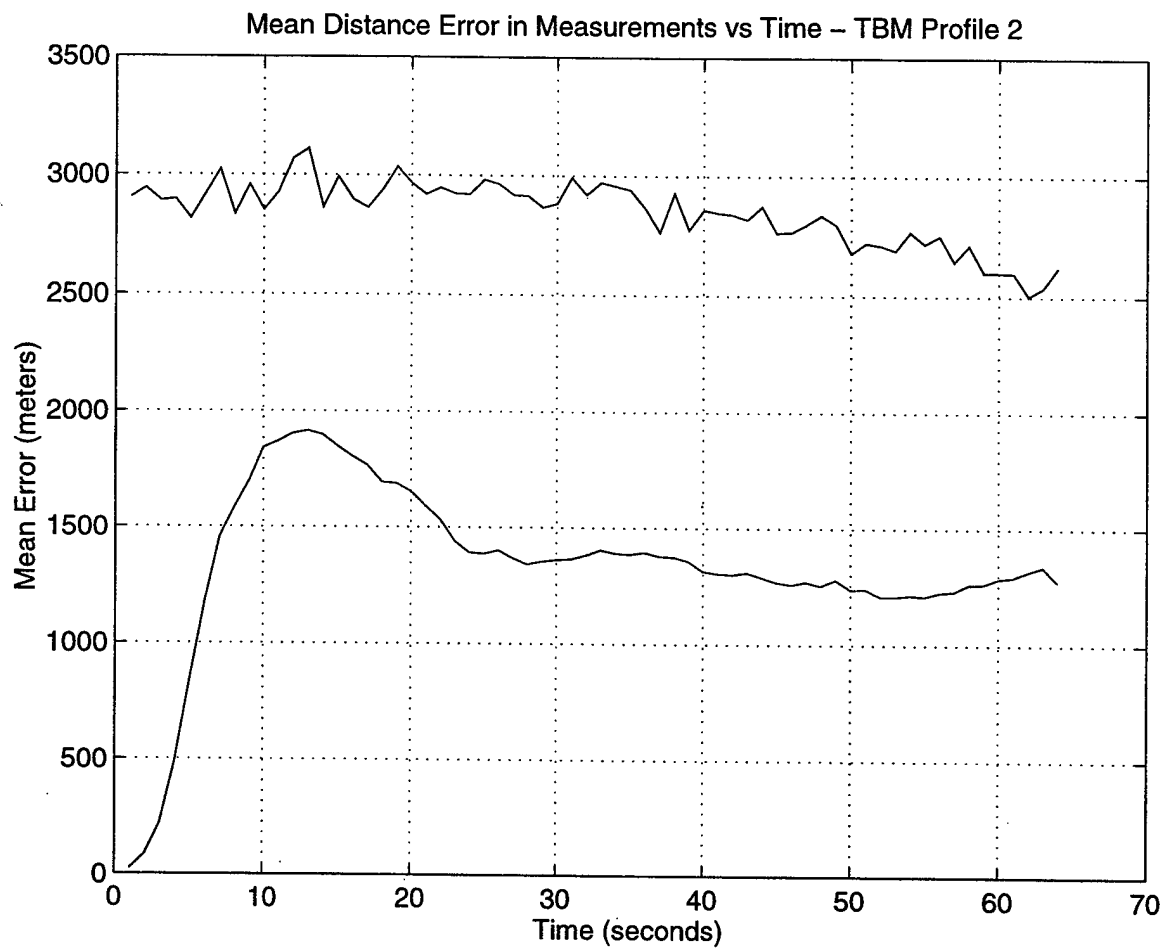


TBM Trajectory (Profile 2) with Measurement Noise, 100 Runs.

TBM Profile 2 and EKF(accel model)Trajectory

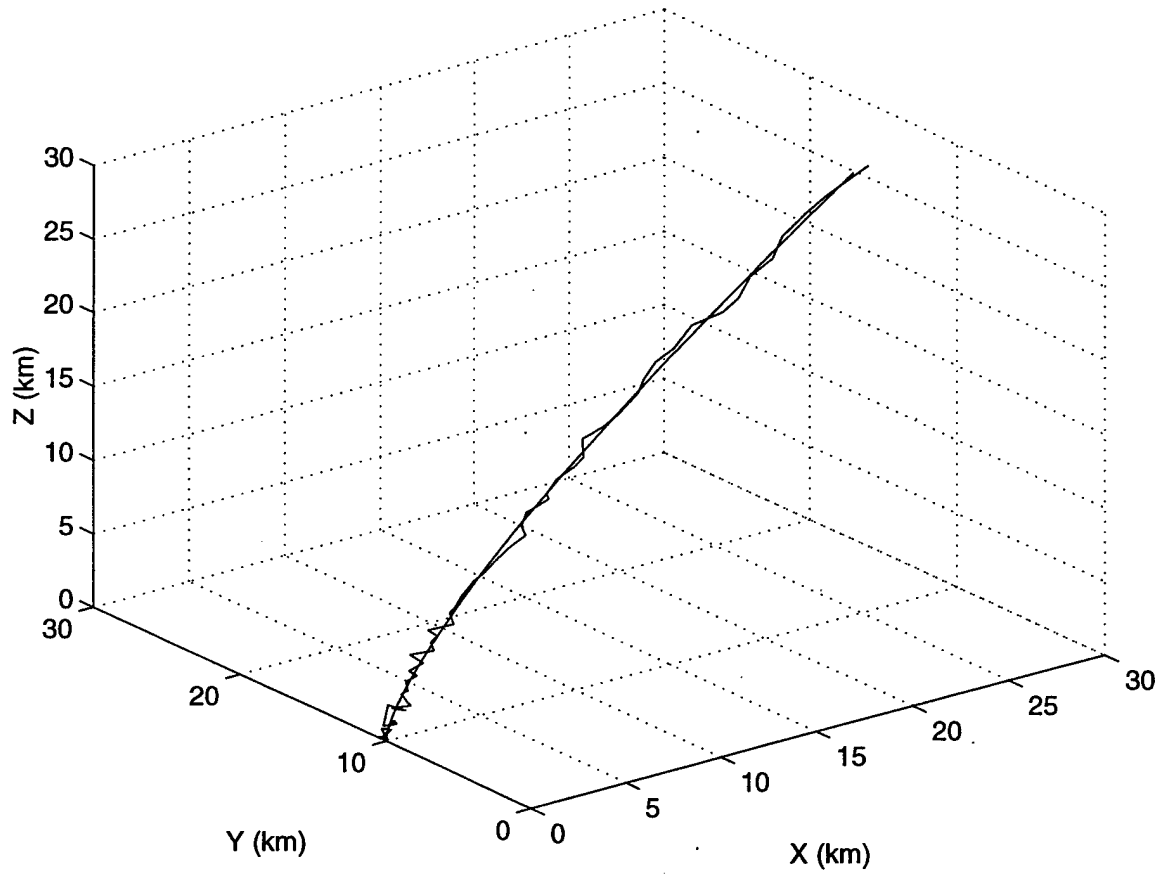


TBM Trajectory (Profile 2) and EKF Trajectory, 100 Runs.



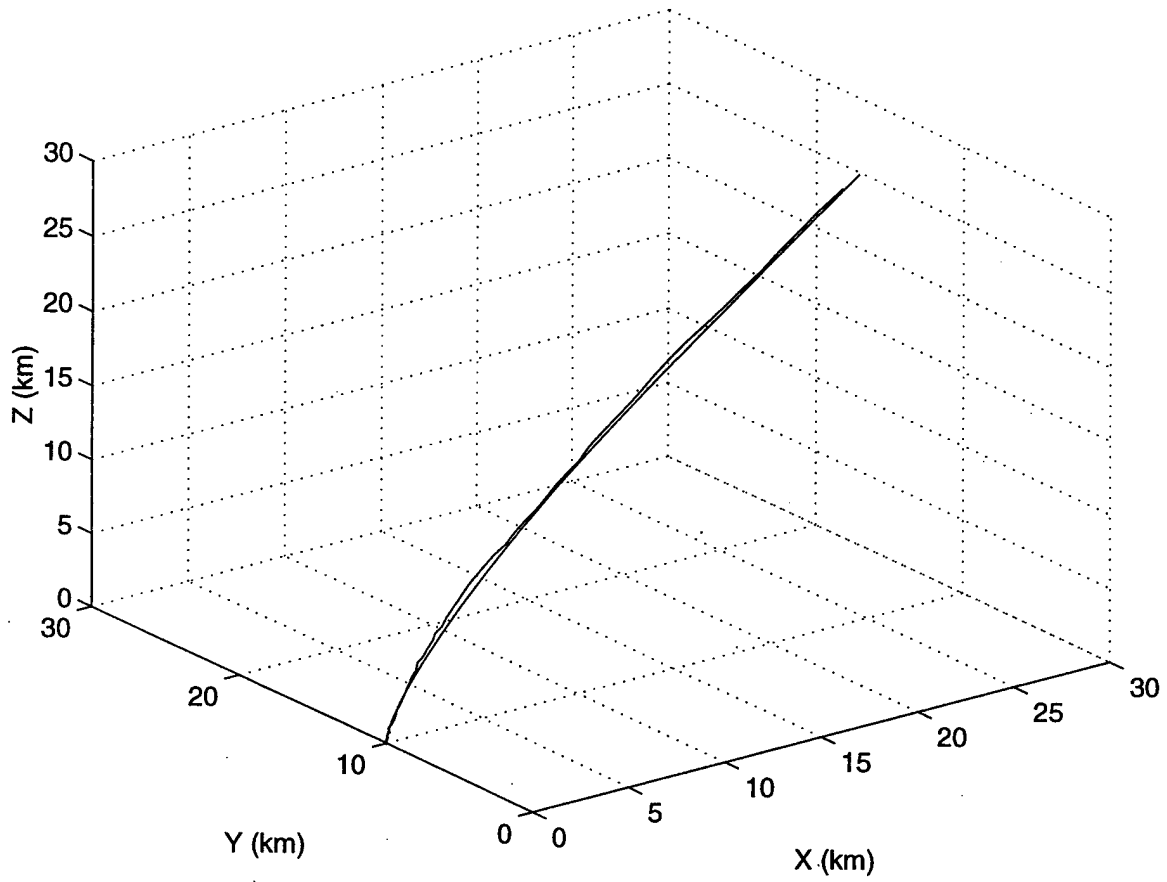
EKF (Profile 2) Mean Distance Error, 500 Runs.

TBM Profile 2 w/ Measurement Noise

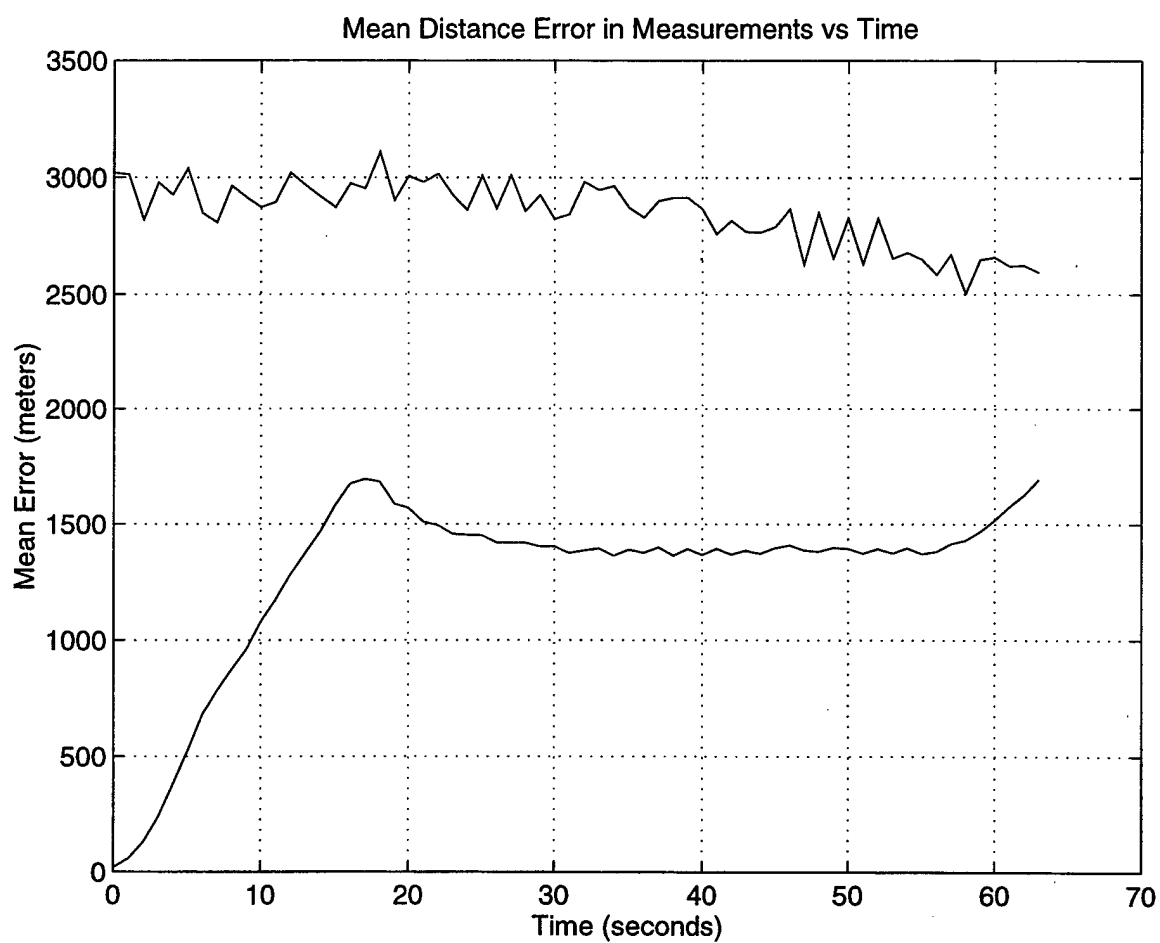


TBM Trajectory (Profile 2) with Measurement Noise, 100 Runs.

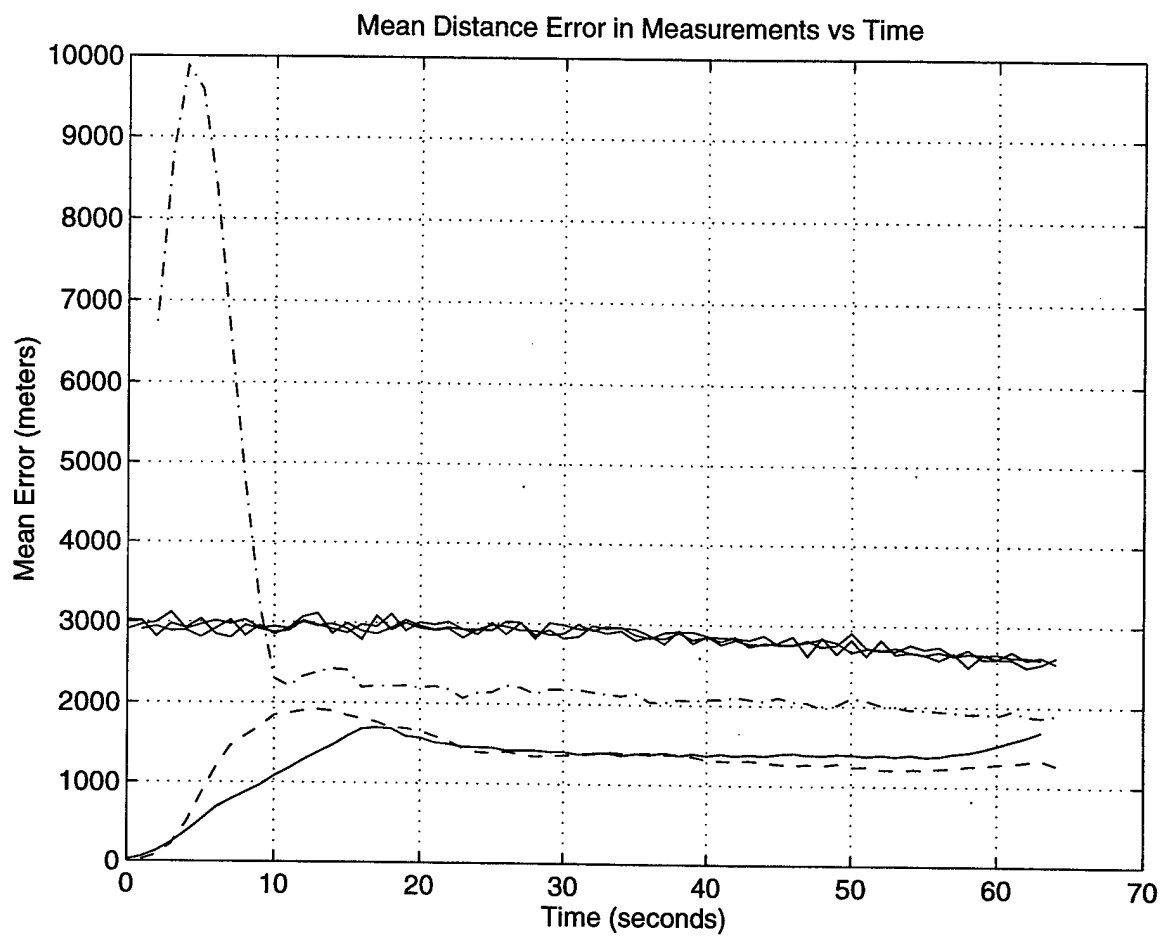
TBM Profile 2 w/ IMM Trajectory



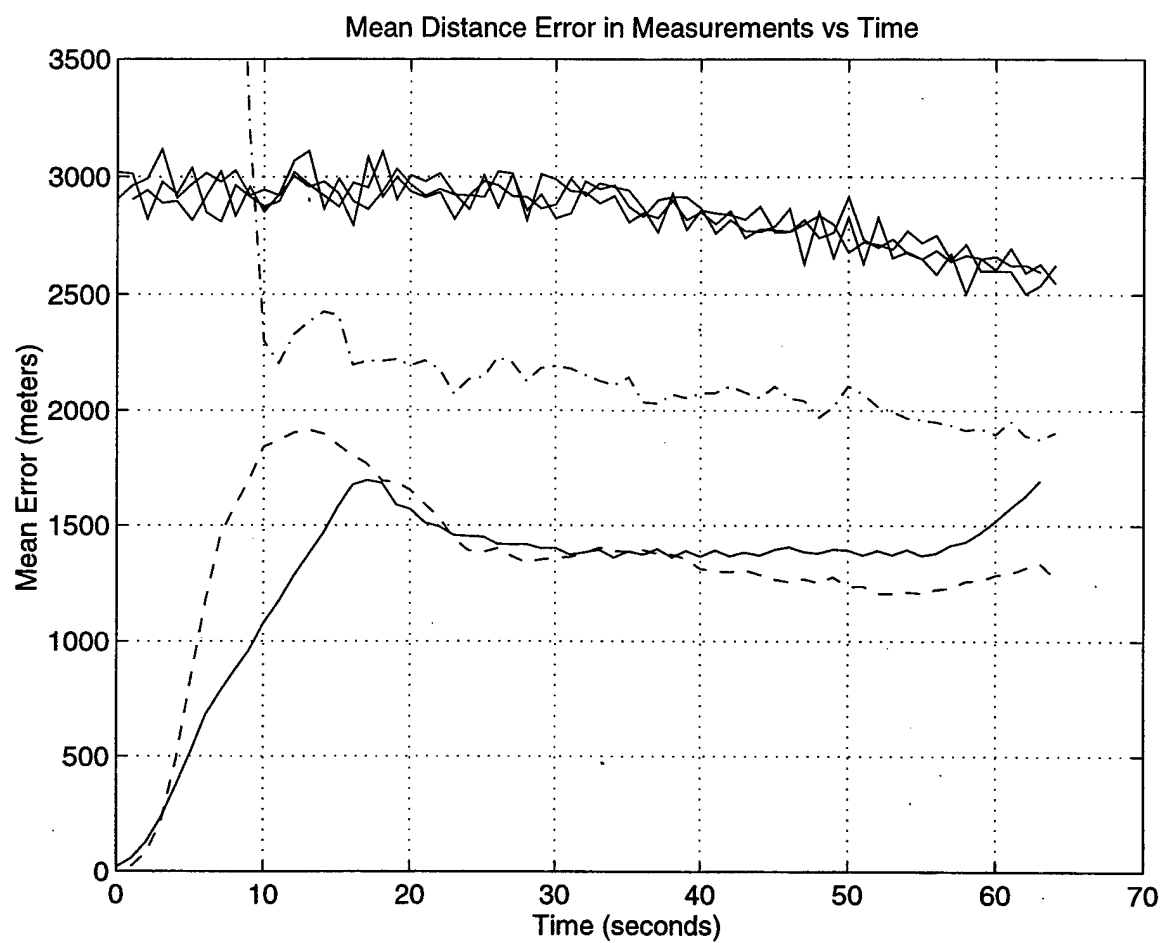
TBM Trajectory (Profile 2) and IMM Trajectory, 100 Runs.



IMM (Profile 2) Mean Distance Error, 500 Runs.

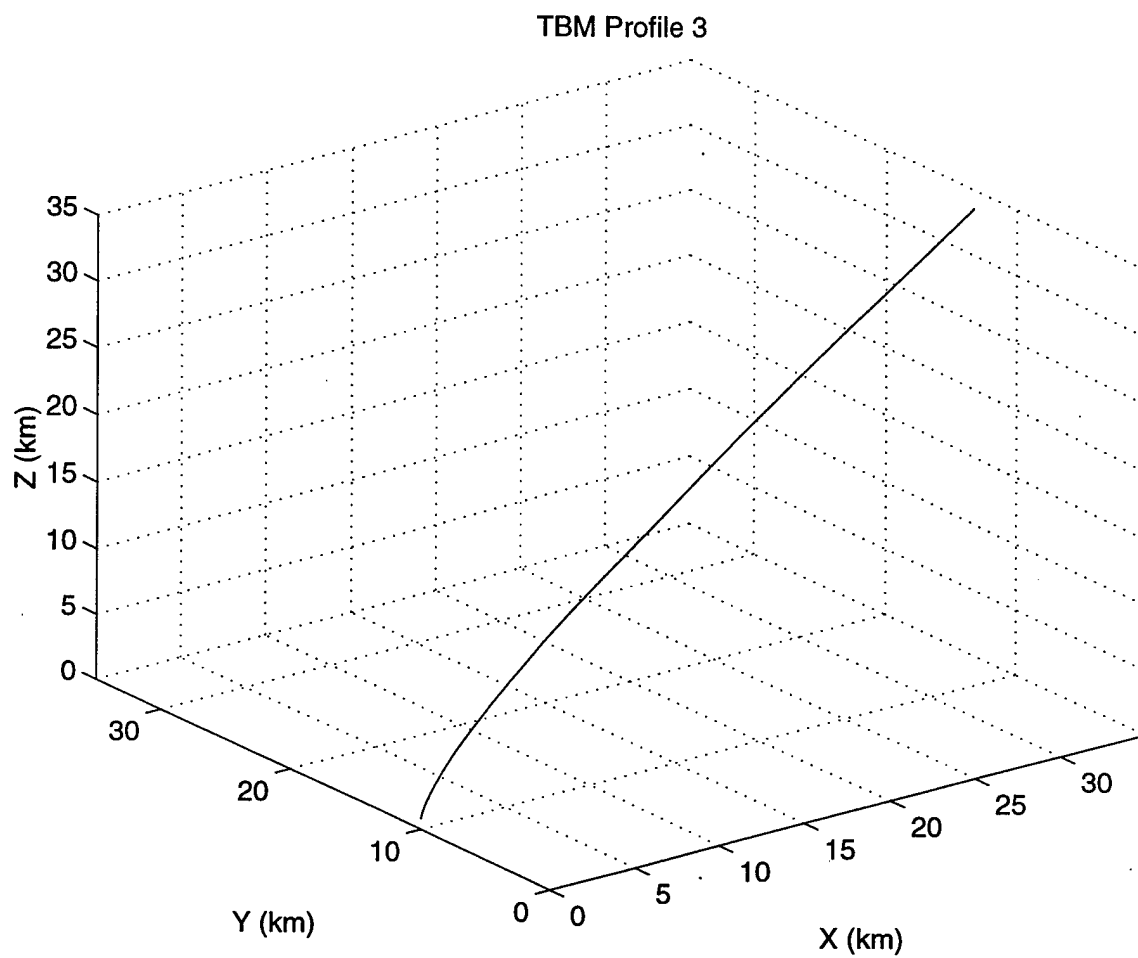


Comparison of α - β - γ , EKF and IMM Mean Distance Error, 500 Runs.



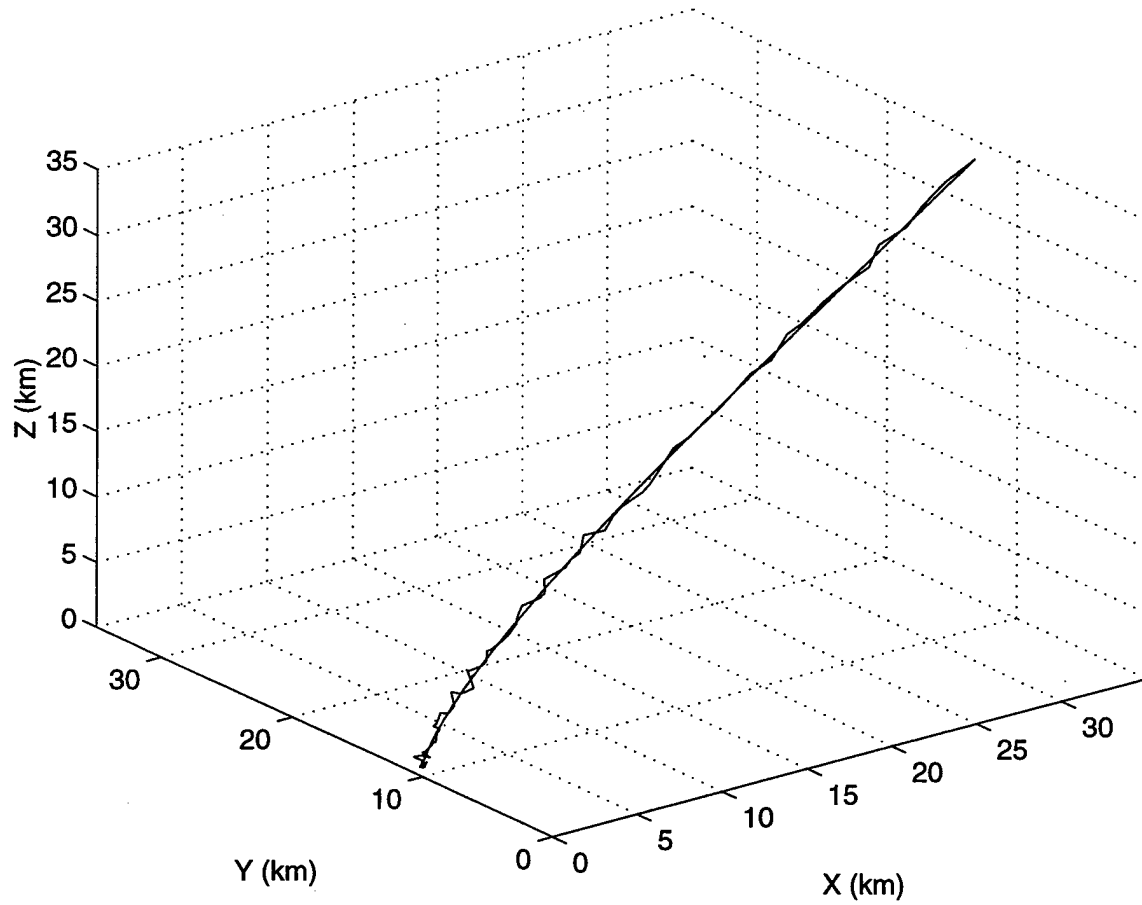
Comparison (Close-up) of Mean Distance Error, 500 Runs.

G. TBM PROFILE 3 ANALYSIS



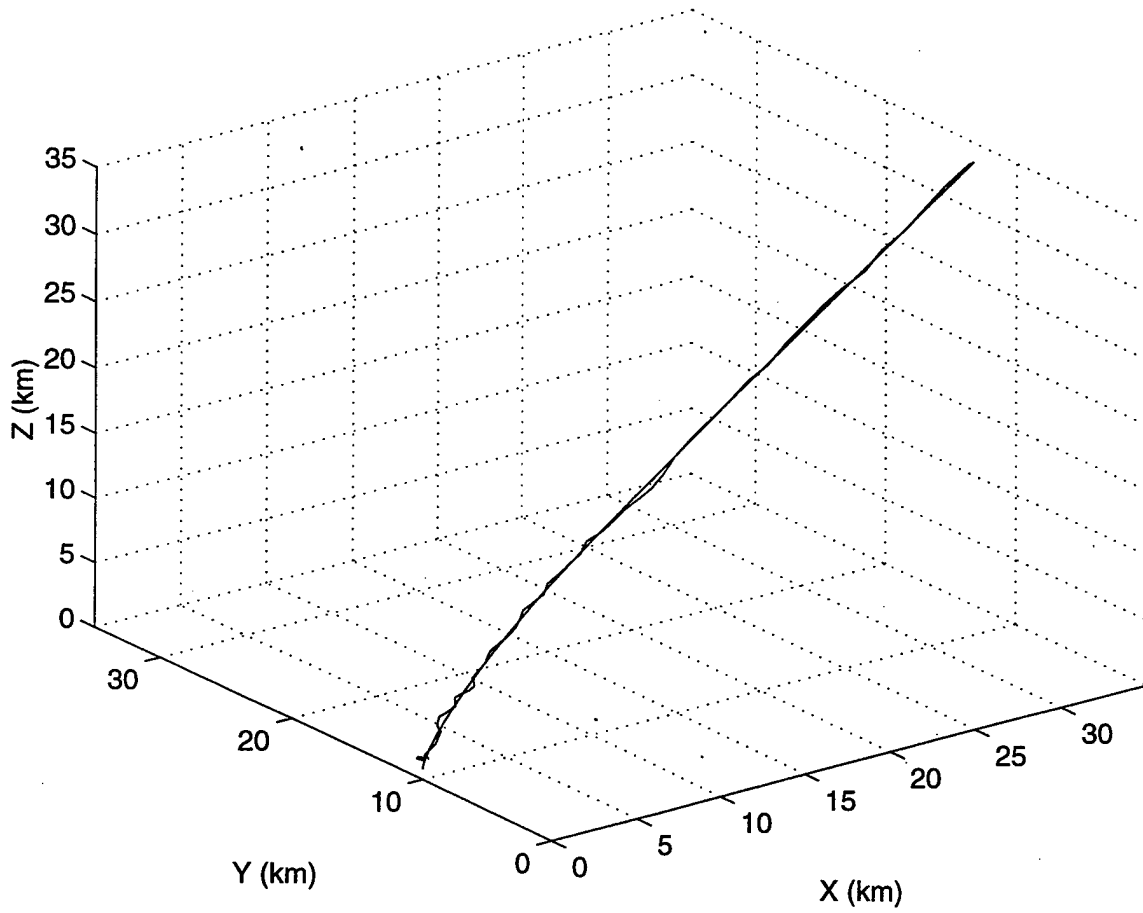
TBM Trajectory (Profile 3).

TBM Profile 3 w/ Measurement Noise

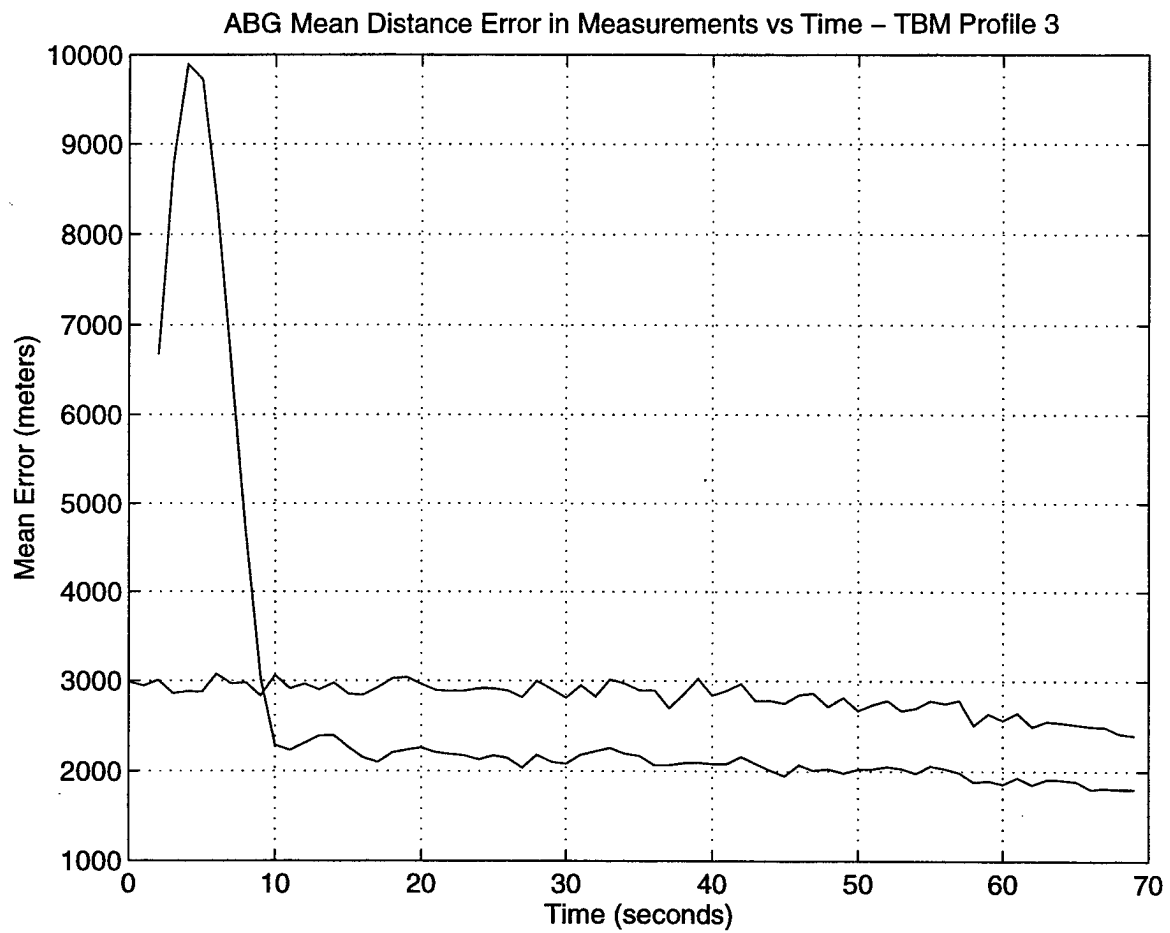


TBM Trajectory (Profile 3) with Measurement Noise, 100 Runs.

TBM Profile 3 and ABG Trajectory

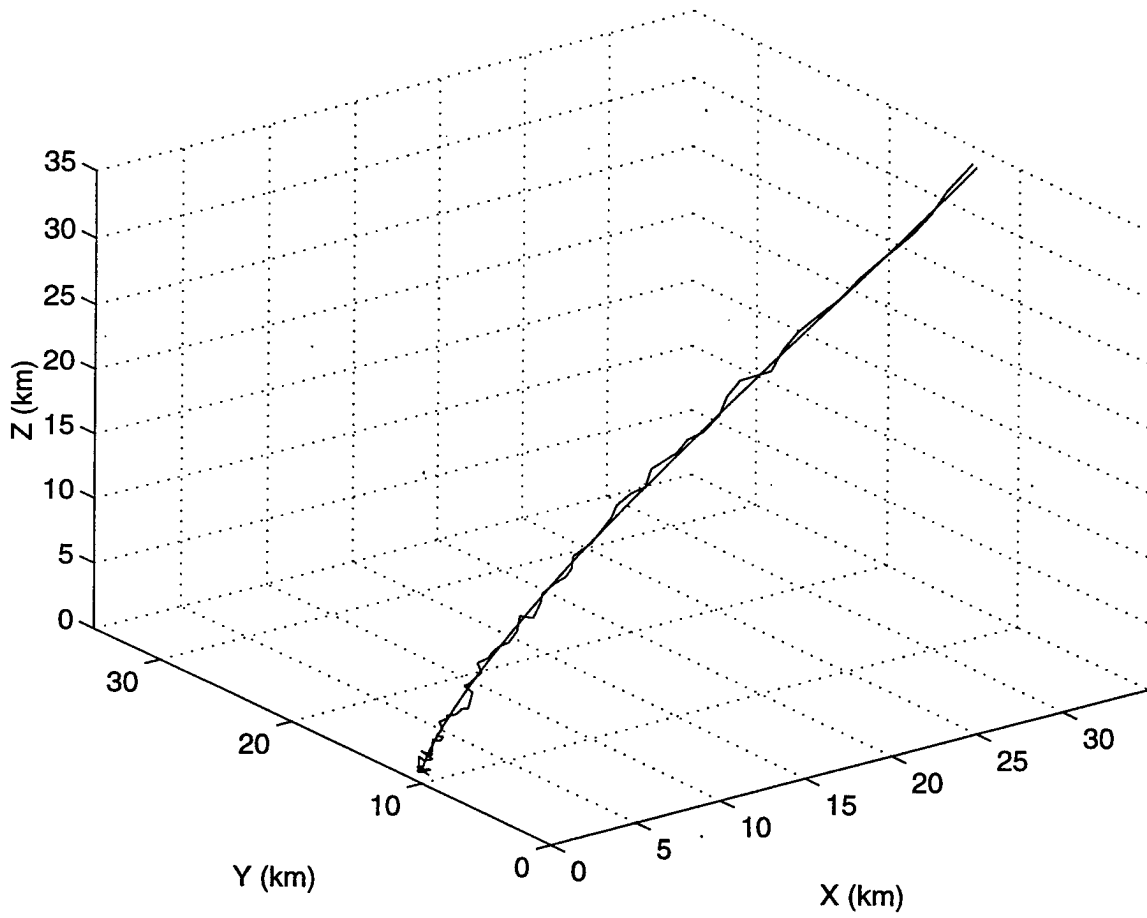


TBM Trajectory (Profile 3) and α - β - γ Trajectory, $\alpha=0.6$, 100 Runs.



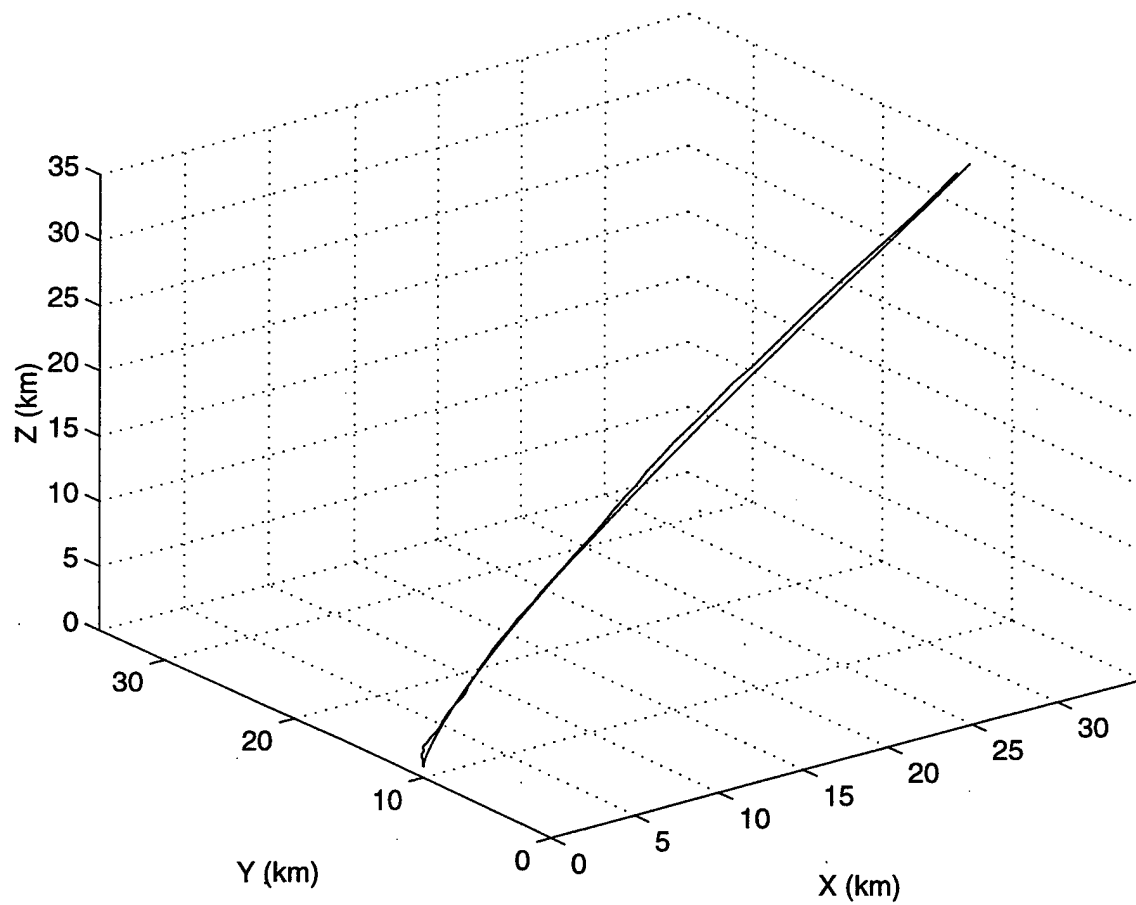
α - β - γ Tracker (Profile 3) Mean Distance Error, $\alpha=0.6$, 500 Runs.

TBM Profile 3 w/ Measurement Noise

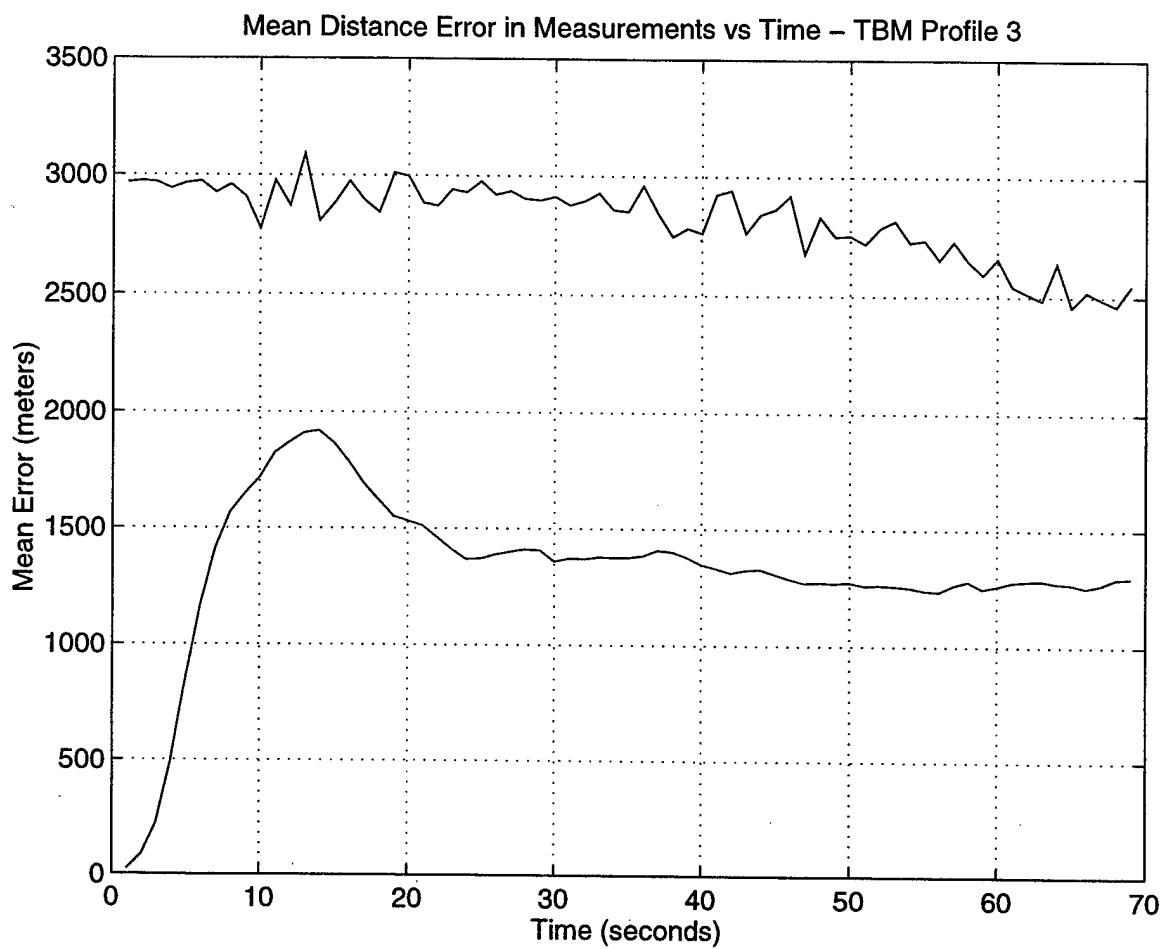


TBM Trajectory (Profile 3) with Measurement Noise, 100 Runs.

TBM Profile 3 and EKF(accel model)Trajectory

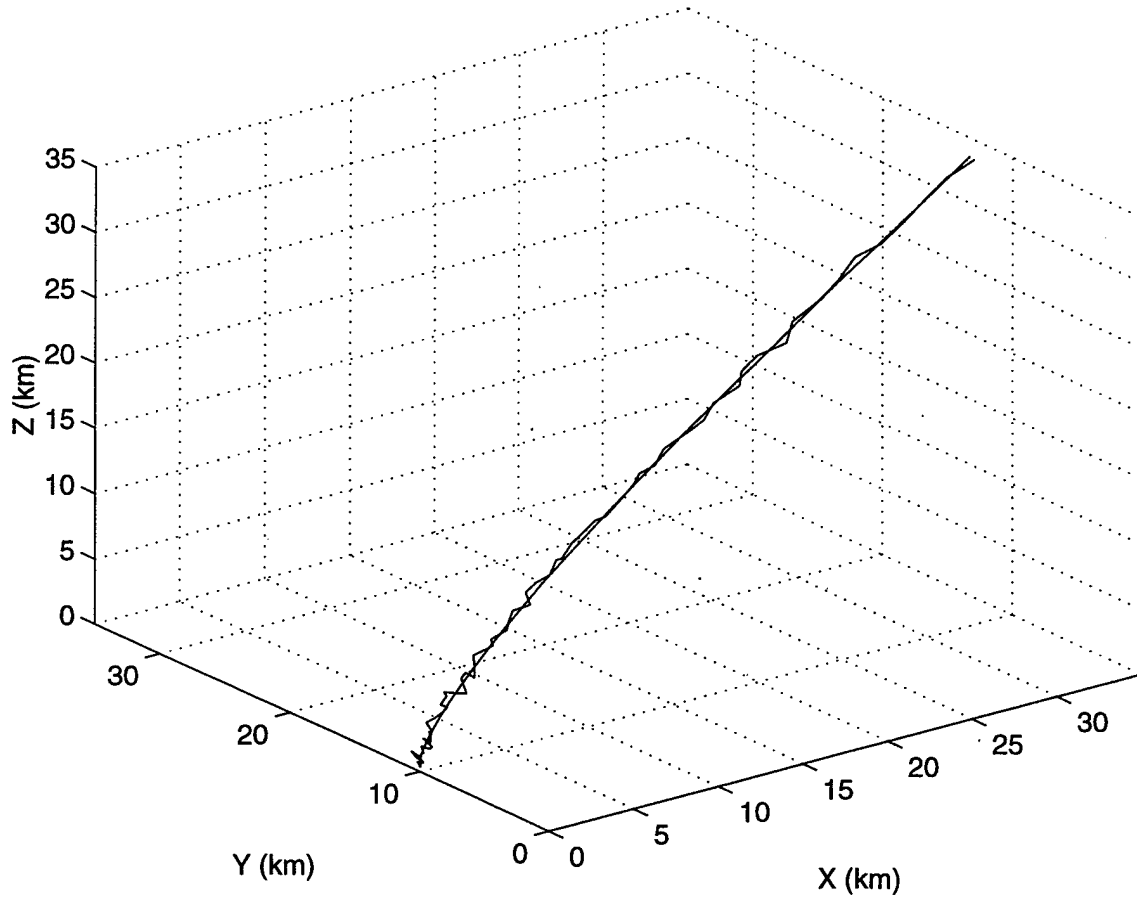


TBM Trajectory (Profile 3) and EKF Trajectory, 100 Runs.



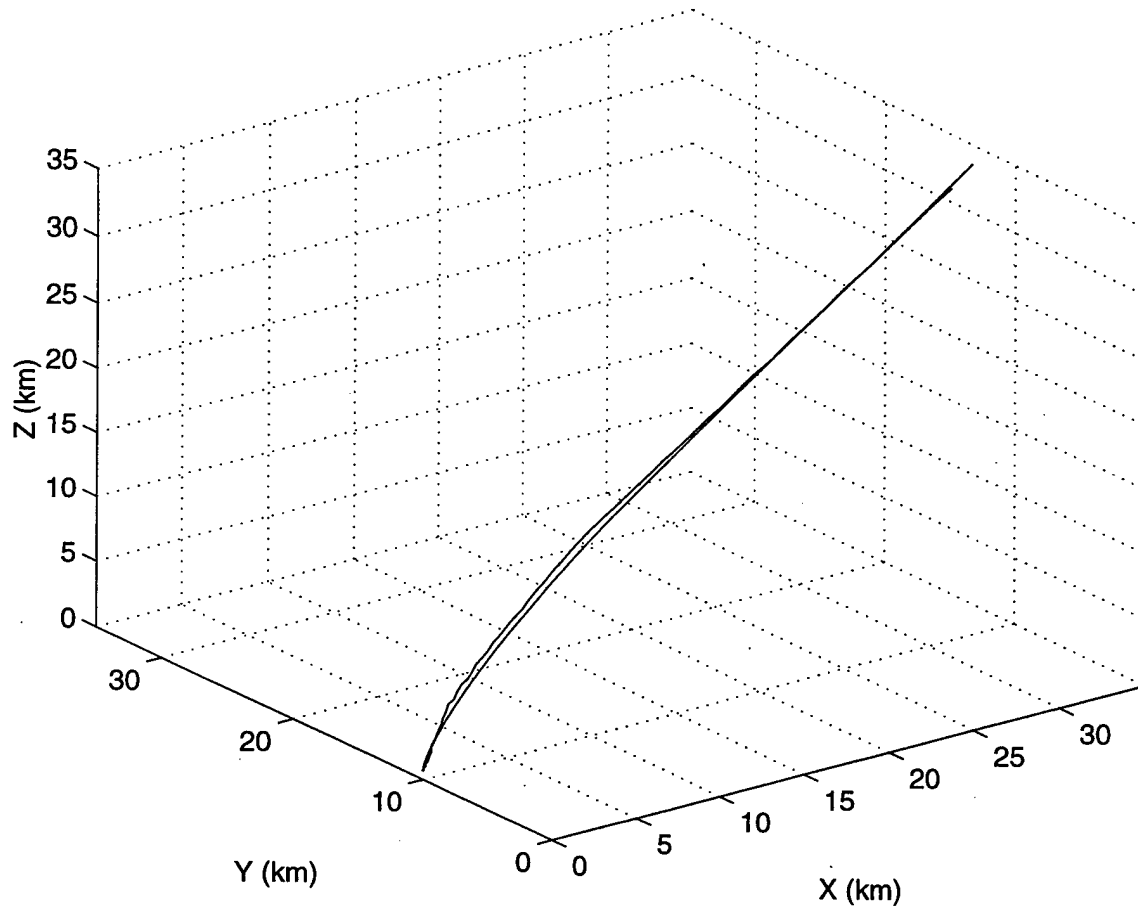
EKF (Profile 3) Mean Distance Error, 500 Runs.

TBM Profile 3 w/ Measurement Noise

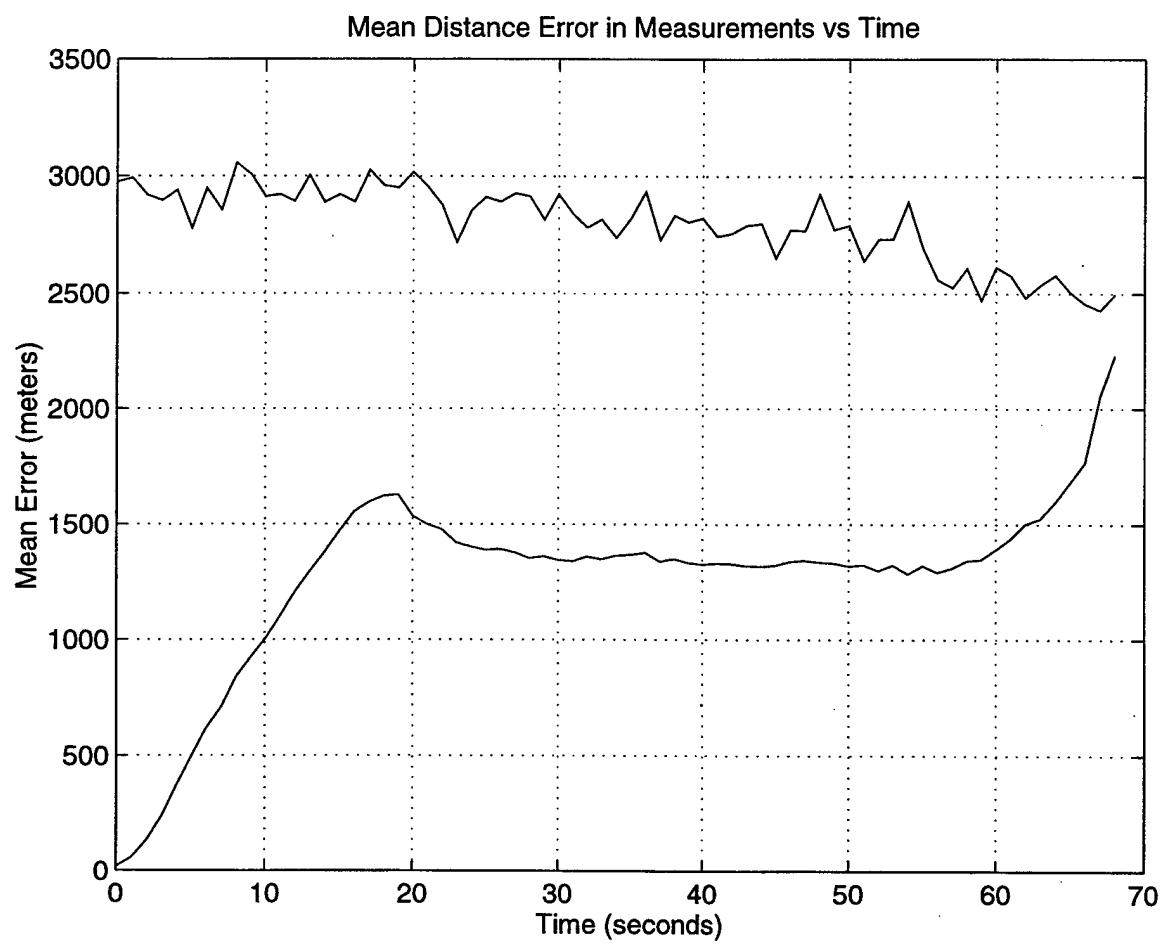


TBM Trajectory (Profile 3) with Measurement Noise, 100 Runs.

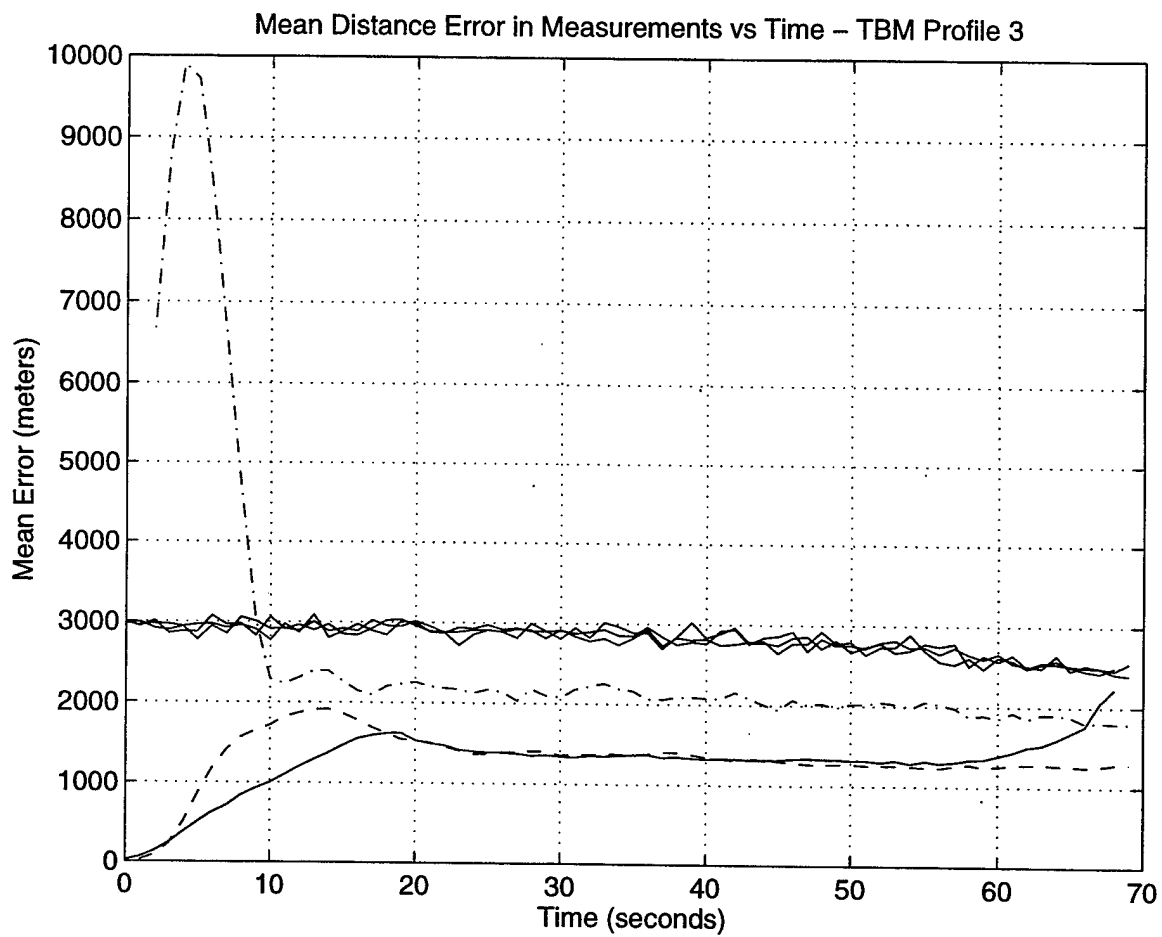
TBM Profile 3 w/ IMM Trajectory



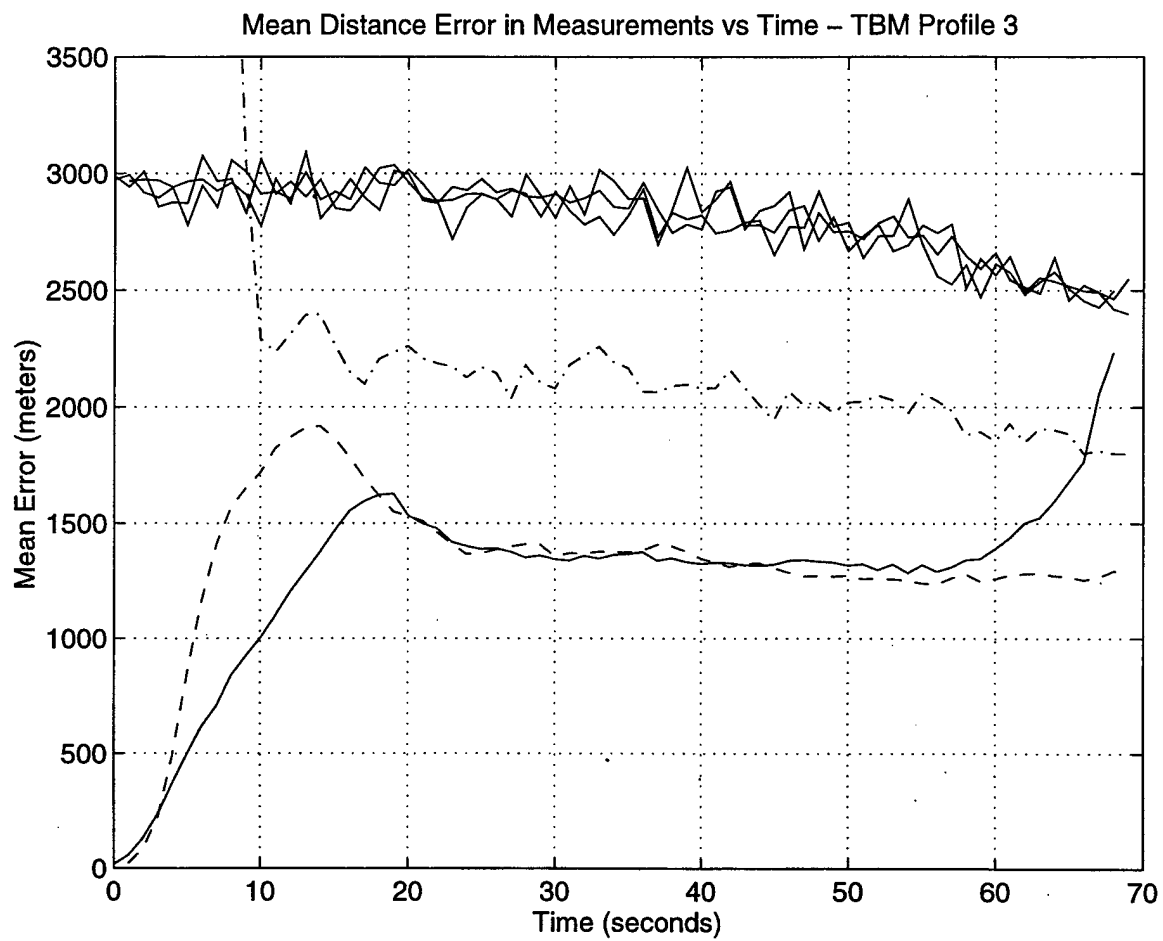
TBM Trajectory (Profile 3) and IMM Trajectory, 100 Runs.



IMM (Profile 3) Mean Distance Error, 500 Runs.



Comparison of α - β - γ , EKF and IMM Mean Distance Error, 500 Runs.



Comparison (Close-up) of Mean Distance Error, 500 Runs.

H. SOURCE CODE FOR α - β - γ , EKF AND IMM ALGORITHMS ON TBM

PROFILE DATA

```
%*****
% tbmdat.m
% LT Tony San Jose
% Thesis Advisor: R.G. Hutchins
% 21FEB98
%
% This program stores TBM profiles 1-5 into missilevec data for use
% in tracking algorithms.
%*****
% TBM Profile Number 1

tbmdat1 = [0.00  136.26  0.0000  0.0000;
  1.00  136.26  0.006  0.0000;
  2.00  136.26  0.026  0.000;
  3.00  136.26  0.058  0.000;
  4.00  136.26  0.103  0.000;
  5.00  136.26  0.1636  0.001;
  6.00  136.26  0.235  0.004;
  7.00  136.26  0.322  0.0100;
  8.00  136.26  0.423  0.020;
  9.00  136.26  0.537  0.036;
  10.00 136.26  0.666  0.058;
  11.00 136.26  0.809  0.087;
  12.00 136.26  0.965  0.124;
  13.00 136.26  1.136  0.171;
  14.00 136.26  1.3217 0.226;
  15.00 136.26  1.52  0.292;
  16.00 136.26  1.733  0.367;
  17.00 136.26  1.962  0.453;
  18.00 136.26  2.204  0.550;
  19.00 136.26  2.460  0.658;
  20.00 136.26  2.731  0.777;
  21.00 136.26  3.015  0.908;
  22.00 136.26  3.312  1.050;
  23.00 136.26  3.623  1.205;
  24.00 136.26  3.948  1.372;
  25.00 136.26  4.286  1.551;
  26.00 136.26  4.637  1.744;
  27.00 136.26  5.001  1.950;
  28.00 136.26  5.378  2.17;
  29.00 136.26  5.769  2.404;
  30.00 136.26  6.174  2.652;
  31.00 136.26  6.591  2.916;
  32.00 136.26  7.023  3.195;
  33.00 136.26  7.469  3.491;
  34.00 136.26  7.928  3.803;
  35.00 136.26  8.402  4.1320;
  36.00 136.26  8.890  4.4790;
```

37.00	136.26	9.393	4.844;
38.00	136.26	9.911	5.229;
39.00	136.26	10.444	5.633;
40.00	136.26	10.992	6.057;
41.00	136.14	11.556	6.502;
42.00	136.00	12.136	6.969;
43.00	135.86	12.732	7.459;
44.00	135.72	13.345	7.973;
45.00	135.58	13.975	8.5110;
46.00	135.44	14.622	9.075;
47.00	135.30	15.288	9.665;
48.00	135.16	15.972	10.282;
49.00	135.02	16.675	10.928;
50.00	134.88	17.397	11.604;
51.00	134.74	18.140	12.309;
52.00	134.60	18.904	13.045;
53.00	134.46	19.690	13.813;
54.00	134.32	20.499	14.613;
55.00	134.18	21.332	15.446;
56.00	133.43	22.190	16.3140;
57.00	130.50	23.075	17.217;
58.00	127.00	23.986	18.155;
59.00	121.00	24.925	19.131;
60.00	111.00	25.894	20.145;
61.00	86.00	26.894	21.199;
62.00	65.00	27.925	22.293];

% TBM Profile Number 2

tbmdat2=[0.00 136.26 0.0000 0.0000;			
1.00	136.26	0.0064	0.0000;
2.00	136.26	0.0256	0.0001;
3.00	136.26	0.0579	0.0002;
4.00	136.26	0.1035	0.0001;
5.00	136.26	0.1626	0.0009;
6.00	136.26	0.2355	0.0039;
7.00	136.26	0.3222	0.0100;
8.00	136.26	0.4228	0.0203;
9.00	136.26	0.5374	0.0358;
10.00	136.26	0.6661	0.0576;
11.00	136.26	0.8087	0.0868;
12.00	136.26	0.9653	0.1243;
13.00	136.26	1.1359	0.1707;
14.00	136.26	1.3207	0.2264;
15.00	136.26	1.5199	0.2919;
16.00	136.26	1.7335	0.3675;
17.00	136.26	1.9615	0.4535;
18.00	136.26	2.2038	0.5503;
19.00	136.26	2.4602	0.6581;
20.00	136.26	2.7305	0.7771;
21.00	136.26	3.0146	0.9078;
22.00	136.26	3.3123	1.0502;
23.00	136.26	3.6234	1.2047;

24.00	136.26	3.9479	1.3717;
25.00	136.26	4.2856	1.5513;
26.00	136.26	4.6366	1.7439;
27.00	136.26	5.0008	1.9499;
28.00	136.26	5.3784	2.1697;
29.00	136.26	5.7692	2.4037;
30.00	136.26	6.1736	2.6524;
31.00	136.26	6.5915	2.9161;
32.00	136.26	7.0231	3.1954;
33.00	136.26	7.4687	3.4908;
34.00	136.26	7.9283	3.8028;
35.00	136.26	8.4021	4.1320;
36.00	136.26	8.8904	4.4790;
37.00	136.26	9.3933	4.8443;
38.00	136.26	9.9111	5.2287;
39.00	136.26	10.4440	5.6326;
40.00	136.26	10.9922	6.0569;
41.00	136.14	11.5560	6.5022;
42.00	136.00	12.1358	6.9694;
43.00	135.86	12.7319	7.4594;
44.00	135.72	13.3448	7.9729;
45.00	135.58	13.9748	8.5110;
46.00	135.44	14.6224	9.0746;
47.00	135.30	15.2879	9.6647;
48.00	135.16	15.9718	10.2823;
49.00	135.02	16.6746	10.9285;
50.00	134.88	17.3969	11.6039;
51.00	134.74	18.1396	12.3093;
52.00	134.60	18.9036	13.0454;
53.00	134.46	19.6897	13.8131;
54.00	134.32	20.4989	14.6132;
55.00	134.18	21.3321	15.4465;
56.00	133.43	22.1903	16.3140;
57.00	130.50	23.0745	17.2166;
58.00	127.00	23.9859	18.1553;
59.00	121.00	24.9255	19.1312;
60.00	111.00	25.8944	20.1453;
61.00	86.00	26.8938	21.1987;
62.00	65.00	27.9250	22.2926;
63.00	20.00	28.9836	23.4225;
64.00	0.00	30.0367	24.5560];

% TBM Profile Number 3

```
tbmdat3= [0.00    36.40 0.8230    0.0025;
1.00    36.40 0.8291    0.0025;
2.00    36.66 0.8478    0.0026;
3.00    36.66 0.8791    0.0027;
4.00    36.66 0.9231    0.0049;
5.00    36.66 0.9796    0.0124;
6.00    36.92 1.0490    0.0254;
%6.86   36.92 1.1190    0.0408;
7.00    36.92 1.1310    0.0437;
8.00    37.18 1.2260    0.0675;
```

9.00	37.44	1.3350	0.0974;
10.00	37.44	1.4570	0.1338;
11.00	37.70	1.5920	0.1773;
12.00	37.96	1.7400	0.2286;
13.00	38.22	1.9020	0.2881;
14.00	38.74	2.0780	0.3564;
15.00	39.52	2.2670	0.4339;
16.00	40.30	2.4690	0.5211;
17.00	41.34	2.6850	0.6186;
18.00	42.38	2.9140	0.7268;
19.00	43.42	3.1570	0.8462;
20.00	44.46	3.4140	0.9771;
21.00	45.50	3.6840	1.1200;
22.00	46.80	3.9660	1.2750;
23.00	48.10	4.2620	1.4430;
24.00	49.40	4.5710	1.6240;
25.00	50.96	4.8930	1.8180;
26.00	52.26	5.2280	2.0270;
27.00	53.82	5.5760	2.2490;
28.00	55.38	5.9370	2.4860;
29.00	56.94	6.3110	2.7390;
30.00	58.76	6.6980	3.0060;
31.00	60.32	7.0980	3.2900;
32.00	62.14	7.5120	3.5910;
33.00	63.96	7.9400	3.9080;
34.00	65.78	8.3810	4.2430;
35.00	67.86	8.8360	4.5970;
36.00	70.20	9.3060	4.9690;
37.00	72.28	9.7900	5.3600;
38.00	74.62	10.2900	5.7700;
39.00	77.48	10.8000	6.2020;
40.00	80.08	11.3300	6.6540;
41.00	82.94	11.8800	7.1280;
42.00	85.80	12.4400	7.6250;
43.00	88.66	13.0100	8.1440;
44.00	91.78	13.6100	8.6880;
45.00	94.64	14.2200	9.2560;
46.00	96.72	14.8500	9.8500;
47.00	98.80	15.5000	10.4700;
48.00	100.88	16.1600	11.1200;
48.72	101.92	16.6600	11.6000;
49.00	102.18	16.8500	11.7900;
50.00	103.48	17.5600	12.5000;
51.00	104.52	18.2800	13.2300;
52.00	105.56	19.0300	14.0000;
53.00	106.60	19.8000	14.7900;
54.00	107.38	20.5900	15.6200;
55.00	108.42	21.4100	16.4800;
56.00	109.20	22.2500	17.3800;
57.00	109.72	23.1200	18.3100;
58.00	109.98	24.0200	19.2700;
59.00	98.28	24.9400	20.2700;
60.00	86.32	25.8900	21.3100;
61.00	52.26	26.8700	22.3800;

62.00	14.12	27.8900	23.5000;	
63.00	8.11	28.9300	24.6500;	
64.00	6.08	30.0100	25.8500;	
65.00	5.93	31.1200	27.0900;	
66.00	5.80	32.2700	28.3700;	
67.00	5.80	33.4600	29.6900;	
68.00	5.80	34.6800	31.0700;	
69.00	5.80	35.9500	32.4900];	
%	69.46	0.00	36.5500	33.1600];

% TBM Profile Number 4

```
tbmdat4 =[0.00    36.66 0.0000    0.0000;
1.00    36.66 0.0030    0.0000;
2.00    36.66 0.0119    0.0000;
3.00    36.66 0.0270    0.0001;
4.00    36.66 0.0483    0.0003;
5.00    36.66 0.0760    0.0008;
6.00    36.66 0.1101    0.0018;
7.00    36.66 0.1508    0.0032;
8.00    36.66 0.1981    0.0052;
9.00    36.66 0.2523    0.0080;
10.00   36.66 0.3133    0.0118;
11.00   36.66 0.3814    0.0166;
12.00   36.92 0.4567    0.0226;
13.00   36.92 0.5391    0.0302;
14.00   36.92 0.6289    0.0393;
15.00   36.92 0.7262    0.0502;
16.00   37.18 0.8310    0.0632;
17.00   37.18 0.9435    0.0784;
18.00   37.44 1.0638    0.0961;
19.00   37.70 1.1919    0.1164;
20.00   37.96 1.3281    0.1396;
21.00   38.22 1.4723    0.1659;
22.00   38.74 1.6247    0.1956;
23.00   39.26 1.7854    0.2289;
24.00   39.78 1.9545    0.2660;
25.00   40.56 2.1322    0.3073;
26.00   41.34 2.3185    0.3529;
27.00   42.38 2.5135    0.4031;
28.00   43.16 2.7174    0.4582;
29.00   44.20 2.9302    0.5184;
30.00   45.24 3.1522    0.5840;
31.00   46.28 3.3833    0.6553;
32.00   47.32 3.6237    0.7326;
33.00   48.36 3.8734    0.8161;
34.00   49.66 4.1326    0.9061;
35.00   50.96 4.4014    1.0029;
36.00   52.26 4.6796    1.1067;
37.00   53.56 4.9673    1.2177;
38.00   54.86 5.2641    1.3360;
39.00   56.16 5.5698    1.4619;
40.00   57.72 5.8846    1.5956;
41.00   59.28 6.2082    1.7373;
```

42.00	60.58	6.5409	1.8872;
43.00	62.14	6.8826	2.0457;
44.00	63.70	7.2335	2.2130;
45.00	65.26	7.5939	2.3895;
46.00	66.82	7.9640	2.5756;
47.00	68.64	8.3440	2.7716;
48.00	70.46	8.7341	2.9779;
49.00	72.28	9.1344	3.1949;
50.00	74.36	9.5452	3.4228;
51.00	76.44	9.9667	3.6621;
52.00	78.78	10.3990	3.9132;
53.00	81.12	10.8430	4.1764;
54.00	83.72	11.2980	4.4521;
55.00	86.32	11.7640	4.7409;
56.00	88.92	12.2430	5.0430;
57.00	91.52	12.7330	5.3589;
58.00	94.12	13.2360	5.6891;
59.00	96.98	13.7520	6.0339;
60.00	99.84	14.2800	6.3938;
61.00	101.92	14.8220	6.7693;
62.00	103.74	15.3760	7.1607;
63.00	105.30	15.9450	7.5686;
64.00	106.60	16.5270	7.9933;
65.00	107.90	17.1240	8.4354;
66.00	108.94	17.7350	8.8954;
67.00	109.72	18.3600	9.3738;
68.00	110.76	19.0010	9.8710;
69.00	111.54	19.6570	10.3880;
70.00	112.32	20.3290	10.9240;
71.00	113.10	21.0170	11.4810;
72.00	114.14	21.7220	12.0590;
73.00	114.92	22.4430	12.6590;
74.00	115.44	23.1810	13.2800;
75.00	115.96	23.9370	13.9240;
76.00	112.06	24.7100	14.5920;
77.00	100.62	25.5010	15.2830;
78.00	81.90	26.3110	15.9980;
79.00	39.52	27.1390	16.7390;
80.00	10.40	27.9870	17.5050;
81.00	7.59	28.8550	18.2970;
82.00	6.45	29.7420	19.1160;
83.00	6.24	30.6500	19.9620;
84.00	6.11	31.5780	20.8360;
85.00	6.08	32.5280	21.7390;
86.00	6.08	33.5000	22.6720;
87.00	6.08	34.4930	23.6340;
88.00	6.08	35.5100	24.6270;
89.00	6.08	36.5510	25.6510;
90.00	6.08	37.6160	26.7070;
91.00	6.08	38.7060	27.7950];
% 91.50	6.08	39.2610	28.3520

% TBM Profile Number 5

```
tbmdat5 = [0.00  136.26    0.0000    0.0000;
  1.00  136.26    0.0054    0.0000;
  2.00  136.26    0.0217    0.0001;
  3.00  136.26    0.0491    0.0002;
  4.00  136.26    0.0879    0.0003;
  5.00  136.26    0.1381    0.0012;
  6.00  136.26    0.1997    0.0053;
  7.00  136.26    0.2731    0.0130;
  8.00  136.26    0.3582    0.0246;
  9.00  136.26    0.4551    0.0409;
 10.00  136.26    0.5640    0.0623;
 11.00  136.26    0.6851    0.0894;
 12.00  136.26    0.8183    0.1225;
 13.00  136.26    0.9638    0.1624;
 14.00  136.26    1.1216    0.2094;
 15.00  136.26    1.2920    0.2641;
 16.00  136.26    1.4749    0.3269;
 17.00  136.26    1.6705    0.3984;
 18.00  136.26    1.8787    0.4790;
 19.00  136.26    2.0996    0.5691;
 20.00  136.26    2.3330    0.6692;
 21.00  136.26    2.5788    0.7796;
 22.00  136.26    2.8366    0.9006;
 23.00  136.26    3.1065    1.0325;
 24.00  136.26    3.3881    1.1758;
 25.00  136.26    3.6814    1.3308;
 26.00  136.26    3.9862    1.4976;
 27.00  136.26    4.3024    1.6768;
 28.00  136.26    4.6298    1.8685;
 29.00  136.26    4.9685    2.0733;
 30.00  136.26    5.3183    2.2914;
 31.00  136.26    5.6794    2.5232;
 32.00  136.26    6.0517    2.7692;
 33.00  136.26    6.4353    3.0297;
 34.00  136.26    6.8303    3.3053;
 35.00  136.26    7.2367    3.5962;
 36.00  136.26    7.6546    3.9031;
 37.00  136.26    8.0843    4.2263;
 38.00  136.26    8.5257    4.5663;
 39.00  136.26    8.9791    4.9237;
 40.00  136.26    9.4446    5.2990;
 41.00  136.14    9.9225    5.6926;
 42.00  136.00   10.4127    6.1051;
 43.00  135.86   10.9156    6.5370;
 44.00  135.72   11.4315    6.9889;
 45.00  135.58   11.9604    7.4615;
 46.00  135.44   12.5029    7.9554;
 47.00  135.30   13.0591    8.4713;
 48.00  135.16   13.6294    9.0098;
 49.00  135.02   14.2143    9.5718;
 50.00  134.88   14.8140   10.1580;
 51.00  134.74   15.4290   10.7689;
```


52.00	134.60	16.0600	11.4053;
53.00	134.46	16.7076	12.0676;
54.00	134.32	17.3724	12.7565;
55.00	134.18	18.0552	13.4725;
56.00	133.43	18.7565	14.2162;
57.00	130.50	19.4772	14.9884;
58.00	127.00	20.2178	15.7896;
59.00	121.00	20.9793	16.6207;
60.00	111.00	21.7624	17.4823;
61.00	100.00	22.5678	18.3753;
62.00	85.00	23.3965	19.3003;
63.00	62.00	24.2493	20.2583;
64.00	42.00	25.1271	21.2500;
65.00	30.00	26.0308	22.2765;
66.00	22.00	26.9614	23.3385;
67.00	16.00	27.9199	24.4370;
68.00	14.00	28.9074	25.5730;
69.00	12.50	29.9247	26.7476;
70.00	11.00	30.9732	27.9617;
71.00	10.00	32.0539	29.2164;
72.00	9.30	33.1681	30.5130;
73.00	8.60	34.3169	31.8525;
74.00	8.10	35.5018	33.2362;
75.00	7.60	36.7240	34.6653;
76.00	7.10	37.9850	36.1414;
77.00	6.80	39.2863	37.6656;
78.00	6.40	40.6296	39.2395;
79.00	6.10	42.0164	40.8921;
80.00	5.80	43.4485	42.5725;
81.00	0.00	44.9228	44.3028];

save tbm_dat

```

%*****
% tbmininit.m
%
% LT Tony San Jose
% Thesis Advisor: R.G. Hutchins
% 21FEB98
%
% This program stores the TBM profiles entered in tbmdat.m into the
% variable missilevec for use in our tracking algorithms. The TBM
% data was provided provided by JHUAPL.
%*****

load Tbm_dat;

time1 =tbmdat1(:,1);
alt1 = 1000 * tbmdat1(:,3);
rng1 = 1000 * tbmdat1(:,4);

[rows1, cols1] = size(time1);

for i = 1: rows1
    missilevec1(:,i) = [ time1(i); %t
                        rng1(i); %x
                        0; %vx
                        0; %ax
                        10*1000; %y
                        0; %vy
                        0; %ay
                        alt1(i); %z
                        0; %vz
                        0 ]; %az
end; %#1

%*****

time2 =tbmdat2(:,1);
alt2 = 1000 * tbmdat2(:,3);
rng2 = 1000 * tbmdat2(:,4);

[rows2,cols2] = size(time2);

for i = 1: rows2
    missilevec2(:,i) = [ time2(i); %t
                        rng2(i); %x
                        0; %vx
                        0; %ax
                        10*1000; %y
                        0; %vy
                        0; %ay
                        alt2(i); %z
                        0; %vz
                        0 ]; %az
end; %# 2

```

```
%*****
```

```
time3 =tbmdat3(:,1);
alt3 = 1000 * tbmdat3(:,3);
rng3 = 1000 * tbmdat3(:,4);
```

```
[rows3,cols3] = size(time3);
```

```
for i = 1: rows3
    missilevec3(:,i) = [ time3(i);    %t
                        rng3(i);      %x
                        0;             %vx
                        0;             %ax
                        10*1000;      %y
                        0;             %vy
                        0;             %ay
                        alt3(i);      %z
                        0;             %vz
                        0 ];          %az
```

```
end; %# 3
```

```
%*****
```

```
time4 =tbmdat4(:,1);
alt4 = 1000 * tbmdat4(:,3);
rng4 = 1000 * tbmdat4(:,4);
```

```
[rows4,cols4] = size(time4);
```

```
for i = 1: rows4
    missilevec4(:,i) = [ time4(i);    %t
                        rng4(i);      %x
                        0;             %vx
                        0;             %ax
                        10*1000;      %y
                        0;             %vy
                        0;             %ay
                        alt4(i);      %z
                        0;             %vz
                        0 ];          %az
```

```
end; %# 4
```

```
%*****
```

```
time5 =tbmdat5(:,1);
alt5 = 1000 * tbmdat5(:,3);
rng5 = 1000 * tbmdat5(:,4);
```

```
[rows5,cols5] = size(time5);
```

```
for i = 1: rows5
    missilevec5(:,i) = [ time5(i);    %t
                        rng5(i);      %x
```

```

                                0;          %vx
                                0;          %ax
                                10*1000;   %y
                                0;          %vy
                                0;          %ay
                                alt5(i);    %z
                                0;          %vz
                                0  ];       %az
end; %# 5

clear alt1 alt2 alt3 alt4 alt5 cols1 cols2 cols3 cols4 cols5;
clear i rng1 rng2 rng3 rng4 rng5 rows1 rows2 rows3 rows4 rows5;
clear tbmdat1 tbmdat2 tbmdat3 tbmdat4 tbmdat5;
clear time1 time2 time3 time4 time5;

```

```

%*****
% abg_tbm.m
% LT Tony San Jose
% Thesis Advisor: R.G Hutchins
% 03FEB98
%
% This program tests the Alpha-Beta-Gamma tracker on real TBM profiles
%
% delta = 1 sec
% nloops = 100/500
% alpha = 0.6
%*****

% Load simulation workspace
clear all
load tbminit;
missilevec = missilevec3;
prof_num = 3;

% Define the number of simulation loops
nloops = 100;

% Define the sampling interval
delta = 1;
g = 9.8;

% Define the number of samples
[num_rows,num_cols] = size(missilevec3);
nsamples = num_cols;

% Initialize sensor data
Sensor_posit = [ 100 * 1000;      % sensor is 100 km in x
                 100 * 1000;      % sensor is 100 km in y
                 0 * 1000];       % sensor is 100 km in z

sigma_r = 10;                    % 10 meters std dev in range
sigma_b = 1*pi/180;              % 1 degree std dev in azimuth
sigma_e = 1*pi/180;              % 1 degree std dev in elevation

% Define F matrix (TRANSITION MATRIX) for discrete time
% target motion,  $x(k+1) = F(k)*x(k) + G$ 

f_sub = [1, delta, (delta^2)/2;
         0, 1, delta;
         0, 0, 1];

F = [ f_sub, zeros(3), zeros(3);
      zeros(3), f_sub, zeros(3);
      zeros(3), zeros(3), f_sub ];

% Define G matrix
G = -g * [0;
          0;
          0;
          0];

```

```

0;
0;
0;
0;
(delta^2)/2;
delta;
0];

% Define the H matrix (MEASUREMENT MATRIX), assuming that the
% x, y, and z missile positions are observed directly; z(k) = H(k)*x(k)

H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0, 0];

% Define alpha, beta, gamma tracker parameters

alpha = 0.6;
beta = 2*(2-alpha) - 4*sqrt(1-alpha);
gamma = (beta^2)/(2*alpha);
nu = 1;

K_abg = [alpha,          0,          0;
         beta/(nu*delta), 0,          0;
         gamma/((nu*delta)^2), 0,      0;
         0,             alpha,       0;
         0,             beta/(nu*delta), 0;
         0,             gamma/((nu*delta)^2), 0;
         0,             0,           alpha;
         0,             0,           beta/(nu*delta);
         0,             0,           gamma/((nu*delta)^2)];

% Define initialization parameters

d_sub = [ 1, 0, 0, 0, 0, 0, 0;
         3/(2*delta), 0, 0, -2/delta, 0, 0, 1/(2*delta);
         1/(delta^2), 0, 0, -2/(delta^2), 0, 0, 1/delta^2];

D = [d_sub, zeros(3,2);
     zeros(3,1), d_sub, zeros(3,1);
     zeros(3,2), d_sub];

% x_corr = missilevec(2:10,1); % Initialize from truth

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
for kk = 1: nloops
tic
kk
    % define empty output matrices

```

```

% measurement positions (cartesian) w/error
zout_true_n = [];

% distance error between measurement and true position
error_true = [];

% Kalman estimated trajectory
ABG_track = [];

% error between Kalman track and actual track
track_error = [];

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
for ii = 1:nsamples

    % Process the measurement from Sensor

    % True missile position
    ztrue = [missilevec(2,ii);
             missilevec(5,ii);
             missilevec(8,ii)];

    %*****
    % convert current position to polar coordinates and add
    % sensor noise to the position, generating a noisy measurement
    % from the sensor.
    %*****

    % position relative to the sensor
    zrel = ztrue - Sensor_posit;

    r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2); % range
from sensor
    b = atan2(zrel(2), zrel(1)); % bearing
from sensor
    r_prime = sqrt(zrel(1)^2 + zrel(2)^2); % range in
x/y plane
    e = atan2(zrel(3), r_prime); % elevation
from sensor

    % add noise to the measurement
    r_n = r + sigma_r * randn;
    b_n = b + sigma_b * randn;
    e_n = e + sigma_e * randn;

    % measurement in polar + noise
    z_polar_n = [r_n;
                 b_n;
                 e_n];

```

```

% measurement in cartesian coordinates + noise
z_cart_true_n = [r_prime*cos(b_n);
                 r_prime*sin(b_n);
                 r_n*sin(e_n) ] + Sensor_posit;

z_cart_rel_n = [r_prime*cos(b_n);
                r_prime*sin(b_n);
                r_n*sin(e_n) ];

% compute measurement error in cartesian coordinates
zdiff = ztrue - z_cart_true_n;
disterror = sqrt(zdiff'*zdiff);

% Update the measurement array
% true cartesian measurement + error
zout_true_n = [zout_true_n, z_cart_true_n];

% measurement error (between true measurement & true
measurement w/noise)
error_true = [error_true, disterror];

if ii > 2 % For intialization from the first 3 measurements

%*****
% Prediction
%*****

% Initialization using the first 3 measurements
if ii == 3
    x_corr = D * [zout_true_n(:,3);
                  zout_true_n(:,2);
                  zout_true_n(:,1)];
end; %if ii==3

% ABG Filter prediction equations
x_predict = F * x_corr + G;

%*****
% Correction
%*****

% Convert to relative position to compute RBE coord
x_1 = x_predict(1) - Sensor_posit(1);
x_4 = x_predict(4) - Sensor_posit(2);
x_7 = x_predict(7) - Sensor_posit(3);

% Convert prediction to Range, Bearing, Elev coord
r_hat = sqrt(x_1^2 + x_4^2 + x_7^2);
b_hat = atan2(x_4, x_1);
e_hat = atan2(x_7, sqrt(x_1^2 + x_4^2));

```



```

        % Determine expected measurement
        z_cart_exp_rel = [r_hat*cos(b_hat)*cos(e_hat);
                          r_hat*cos(e_hat)*sin(b_hat);
                          r_hat*sin(e_hat)];

        z_cart_exp_true = z_cart_exp_rel + Sensor_posit;

        % Observed minus expected measurements
        % z_tilde_c = z_cart_true_n - z_cart_exp_true;
        z_tilde_c = z_cart_rel_n - z_cart_exp_rel;

        % Correction equations
        x_corr = x_predict + K_abg * z_tilde_c;

        % Alpha-Beta-Gamma track positions and difference
        between ABG and
        % actual track position and actual target position
        zout_ABG_track = H * x_corr;

        track_diff = ztrue - zout_ABG_track;
        track_error = [track_error, sqrt(track_diff'*track_diff)];

        % Update ABG track trajectory array
        ABG_track = [ABG_track, zout_ABG_track];

    end; % if ii>2

end; % for ii = 1:nsamples

%*****

    if kk == 1, % create first output

        zoutmean_true = zout_true_n;
        mean_ABG_track = ABG_track;
        merror_track = track_error;
        merror = error_true;

    else % create output after 1st run

        zoutmean_true = zoutmean_true + zout_true_n;
        mean_ABG_track = mean_ABG_track + ABG_track;
        merror_track = merror_track + track_error;
        merror = merror + error_true;

    end; % if kk ==1, else
toc
end; % for kk = 1:nloops

%*****
% Compute Means
%*****
    zoutmean_true = zoutmean_true/nloops;

```

```

mean_ABG_track = mean_ABG_track/nloops;
merror         = merror/nloops;      % mean error between
                                       % measurement and true position

merror_track = merror_track/nloops;  % mean error between
                                       % EKF estimated position
                                       % and true position

%*****
% Plot results
%*****
figure(1)
    measurement = zoutmean_true/1000;      % convert to km
    ABG          = mean_ABG_track/1000;    % convert to km
    missile_track = missilevec(:,1:nsamples)/1000; % convert to km

    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');
    %Sensor_posit(1)/1000, Sensor_posit(2)/1000,
    Sensor_posit(3)/1000, 'rx');

    axis([0,35,0,35,0,35]); % profile 1,2,3
% axis('equal')
% axis([0,40,0,40,0,40]); % profile 4,5
    title(['TBM Profile ', num2str(prof_num)]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    print -deps abg3a

figure(2)
    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');
    % measurement(1,:), measurement(2,:), measurement(3,:), 'r-'); %,...

    %Sensor_posit(1)/1000, Sensor_posit(2)/1000, Sensor_posit(3)/1000, 'rx');

    % axis([0,35,0,35,0,35]);
% axis([0,40,0,40,0,40]);
    %axis('equal');
    title(['TBM Profile ', num2str(prof_num), ' w/ Measurement Noise']);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
    print -deps abg3b

figure(3)
    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');
    % ABG(1,1:nsamples-2), ABG(2,1:nsamples-2), ABG(3,1:nsamples-2), 'r-',...

    Sensor_posit(1)/1000, Sensor_posit(2)/1000, Sensor_posit(3)/1000, 'rx');

    % axis([0,25,0,25,0,25]);
% axis([0,40,0,40,0,40]);
    axis([0,35,0,35,0,35]);

```

```

    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
    title(['TBM Profile ', num2str(prof_num),' and ABG Trajectory']);
    print -deps abg3c

figure(4)
    time = missilevec(1,:);
    plot(time(1:nsamples), merror,'g-', time(3:nsamples), merror_track,'r-');
    xlabel('Time (seconds)'),ylabel('Mean Error (meters)'),grid;
    title(['ABG Mean Distance Error in Measurements vs Time - TBM Profile ', num2str(prof_num)]);
    % axis([0,70,0,10000])

%print -deps abg3d

save abg1003

```

```

%*****
% acl_tbm.m
%
% LT Tony San Jose
% Thesis Advisor: R.G. Hutchins
% 21FEB98
% delta = 1.0 sec
% nloops = 100/500
% q^2 = 10
% This program stores the TBM profiles entered in tbmdat.m into the
% variable missilevec for use in our tracking algorithms. The TBM
% data was provided provided by JHUAPL.
%*****

% Load simulation workspace
clear all
load tbminit;
missilevec = missilevec3;
prof_num = 3;

% Define the number of simulation loops
nloops = 500;

% Define the sampling interval
delta = 1;
g = 9.8;

% Define the number of samples
[num_rows,num_cols] = size(missilevec3);
nsamples = num_cols;

% Initialize sensor data
Sensor_posit = [ 100 * 1000;      % sensor is 100 km in x
                 100 * 1000;      % sensor is 100 km in y
                 0 * 1000];       % sensor is 100 km in z

sigma_r = 10;                    % 10 meters std dev in range
sigma_b = 1*pi/180;              % 1 degree std dev in azimuth
sigma_e = 1*pi/180;              % 1 degree std dev in elevation

R = diag([sigma_r^2,              % covariance matrix for uncorrelated
          sigma_b^2,              % range and bearing measurements
          sigma_e^2]);

% Define F matrix (TRANSITION MATRIX) for discrete time
% target motion, x(k+1) = F(k)*x(k) + G

f_sub = [1, delta, (delta^2)/2;
         0, 1, delta;
         0, 0, 1];

F = [ f_sub, zeros(3), zeros(3);

```

```

        zeros(3), f_sub, zeros(3);
        zeros(3), zeros(3), f_sub ];

% Define G matrix
G = -g * [0;
          0;
          0;
          0;
          0;
          0;
          (delta^2)/2;
          delta;
          0];

% Define the H matrix (MEASUREMENT MATRIX), assuming that the
% x, y, and z missile positions are observed directly;

H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 1, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 1, 0, 0];

% Initialize Q, the covariance of the plant noise
% q^2 = scale factor to system noise covariance matrix Q,
% used to account for unmodeled target maneuver acceleration.

q_sqr = 10;

Q_sub = [ (delta^5)/20, (delta^4)/8, (delta^3)/6;
          (delta^4)/8, (delta^3)/3, (delta^2)/2;
          (delta^3)/6, (delta^2)/2, delta ];

Q = q_sqr * [ Q_sub, zeros(3), zeros(3);
              zeros(3), Q_sub, zeros(3);
              zeros(3), zeros(3), Q_sub ];

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
for kk = 1: nloops
tic
kk
    % define empty output matrices

    % measurement positions (cartesian) w/error
    zout_true_n = [];

    % distance error between measurement and true position
    error_true = [];

    % Kalman estimated trajectory
    K_track = [];
    K_accel = [];

```

```

    % error between Kalman track and actual track
    track_error = [];

%*****
% This block is used for the initialization process. Initialize
% from a SWAG.
%*****
    x_swag = missilevec(2:10,1);
    x_swag(9) = 6*g;
    p_swag = eye(9) * 10^4;

    x_corr = x_swag;
    P_corr = p_swag;

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
    for ii = 2:nsamples

        % Process the measurement from Sensor

        % True missile position
        ztrue = [missilevec(2,ii);
                 missilevec(5,ii);
                 missilevec(8,ii)];

%*****
% convert current position to polar coordinates and add
% sensor noise to the position, generating a noisy measurement
% from the sensor.
%*****

        % position relative to the sensor
        zrel = ztrue - Sensor_posit;

        r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2); % range
        from sensor
        b = atan2(zrel(2), zrel(1)); % bearing
        from sensor
        r_prime = sqrt(zrel(1)^2 + zrel(2)^2); % range in
        x/y plane
        e = atan2(zrel(3), r_prime); % elevation
        from sensor

        % add noise to the measurement
        r_n = r + sigma_r * randn;
        b_n = b + sigma_b * randn;
        e_n = e + sigma_e * randn;

        % measurement in polar + noise
        z_polar_n = [r_n;

```

```

        b_n;
        e_n];

% measurement in cartesian coordinates + noise
z_cart_true_n = [r_prime*cos(b_n);
                 r_prime*sin(b_n);
                 r_n*sin(e_n) ] + Sensor_posit;

z_cart_rel_n = [r_prime*cos(b_n);
                r_prime*sin(b_n);
                r_n*sin(e_n) ];

% compute measurement error in cartesian coordinates
zdiff = ztrue - z_cart_true_n;
disterror = sqrt(zdiff'*zdiff);

% Update the measurement array
% true cartesian measurement + error
zout_true_n = [zout_true_n, z_cart_true_n];

% measurement error (between true measurements)
error_true = [error_true, disterror];

%*****
% Prediction
%*****

% Kalman Filter prediction equations
x_predict = F * x_corr + G;
P_predict = F * P_corr * F' + Q;

%*****
% Correction
%*****

% Convert to relative position to compute RBE
coordinates
x_1 = x_predict(1) - Sensor_posit(1);
x_4 = x_predict(4) - Sensor_posit(2);
x_7 = x_predict(7) - Sensor_posit(3);

% Convert prediction to Range, Bearing, Elevation
coordinates
r_hat = sqrt(x_1^2 + x_4^2 + x_7^2);
b_hat = atan2(x_4, x_1);
e_hat = atan2(x_7, sqrt(x_1^2 + x_4^2));

% Determine expected measurement
z_polar_hat = [r_hat;
               b_hat;
               e_hat];

% Observed minus expected measurements

```

```

        z_tilde = z_polar_n - z_polar_hat;

        % The gradient of H evaluated at the most recent estimate
        Hk_r2c1 = -x_4/(x_1^2 + x_4^2);
        Hk_r2c4 = x_1/(x_1^2 + x_4^2);
        Hk_r3c1 = (-x_1*x_7)/( (sqrt(x_1^2 + x_4^2))*(x_1^2 + x_4^2 + x_7^2) );
        Hk_r3c4 = (-x_4*x_7)/( (sqrt(x_1^2 + x_4^2))*(x_1^2 + x_4^2 + x_7^2) );
        Hk_r3c7 = (sqrt(x_1^2 + x_4^2))/(x_1^2 + x_4^2 + x_7^2);

        % Determine H matrix
        Hk = [x_1/r_hat, 0, 0, x_4/r_hat, 0, 0, x_7/r_hat, 0, 0;
              Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
              Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

        % Compute Kalman Gain
        K = P_predict * Hk' * inv(Hk * P_predict * Hk' + R);

        % Correction equations
        x_corr = x_predict + K * z_tilde;
        P_corr = (eye(9) - K*Hk) * P_predict * (eye(9) - K*Hk)' + K*R*K';

        % Kalman track positions and difference between Kalman
and
        % actual track position and actual target position
        zout_K_track = H*x_corr;

        track_diff = ztrue - zout_K_track;
        track_error = [track_error, sqrt(track_diff'*track_diff)];

        % Update KF track trajectory array
        K_track = [K_track, zout_K_track];

        % Estimated accelerations
        accel_out = [x_corr(3,:);
                    x_corr(6,:);
                    x_corr(9,:)];

        % Update KF acceleration array
        K_accel = [K_accel, accel_out];

    end; % for ii = 2:nsamples

%*****

    if kk == 1, % create first output

        zoutmean_true = zout_true_n;
        mean_K_track = K_track;
        merror_track = track_error;
        merror = error_true;

    else % create output after 1st run

```



```

        zoutmean_true = zoutmean_true + zout_true_n;
        mean_K_track = mean_K_track + K_track;
        merror_track = merror_track + track_error;
        merror = merror + error_true;

    end; % if kk ==1, else
toc
end; % for kk = 1:nloops

%*****
% Compute Means
%*****
    zoutmean_true = zoutmean_true/nloops;
    mean_K_track = mean_K_track/nloops;
    merror = merror/nloops; % mean error between
                            % measurement and true position

    merror_track = merror_track/nloops; % mean error between
                                        % EKF estimated position
                                        % and true position

%*****
% Plot results
%*****
figure(1)
    measurement = zoutmean_true/1000; % convert to km
    Kalman_track = mean_K_track/1000; % convert to km
    missile_track = missilevec(:,1:nsamples)/1000; % convert to km

    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');
    %Sensor_posit(1)/1000, Sensor_posit(2)/1000,
    Sensor_posit(3)/1000, 'rx');

    %axis([0,25,0,25,0,25]);
    %axis('equal')
    axis([0,35,0,35,0,35])
    %axis([0,40,0,40,0,40]);
    title(['TBM Profile ', num2str(prof_num)]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
% print -deps ekf3a

figure(2)
plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');
measurement(1,:), measurement(2,:), measurement(3,:), 'r-');
% axis([0,25,0,25,0,25]); % profile 1,2,3,5
% axis([0,40,0,40,0,40]); % profile 4
axis([0,35,0,35,0,35])
%axis('equal');
title(['TBM Profile ', num2str(prof_num), ' w/ Measurement Noise']);
xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'), grid;
% print -deps ekf3b

```

```

figure(3)
    plot3(missile_track(2,1:nsamples), missile_track(5,1:nsamples),
        missile_track(8,1:nsamples),'g-',...
        Kalman_track(1,:), Kalman_track(2,:), Kalman_track(3,:), 'r-');

    %axis([0,25,0,25,0,25]);
    %axis([0,40,0,40,0,40]);
    axis([0,35,0,35,0,35]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
    title(['TBM Profile ', num2str(prof_num),' and EKF(accel
model)Trajectory']);
    %print -deps ekf3c

figure(4)
    time = missilevec(1,:);
    plot(time(2:nsamples), merror,'g-', time(2:nsamples), merror_track,'r-
');
    xlabel('Time (seconds)'),ylabel('Mean Error (meters)'),grid;
    title(['Mean Distance Error in Measurements vs Time - TBM Profile ',
num2str(prof_num)]);
    %axis([0,70,0,10000])
    %print -deps ekf3d

% save ekf5003;
%save ekf1003

```

```

%*****
% imm_tbm.m
%
% LT Tony San Jose
% Thesis Advisor: R.G. Hutchins
% 21FEB98
% q^2 = 10
% nloops = 100/500
% This program stores the TBM profiles entered in tbmdat.m into the
% variable missilevec for use in our tracking algorithms. The TBM
% data was provided provided by JHUAPL.
%*****
% Load simulation workspace
    clear all
    load tbminit;
    missilevec = missilevec1;
    prof_num = 1;

% Define the number of simulation loops
    nloops = 100;

% Define the sampling interval
    delta = 1;
    g = 9.8;

% Define the number of samples
    [num_rows,num_cols] = size(missilevec1);
    nsamples = num_cols;

% Define q^2
    q_sqr_a = 10;
    q_sqr_b = 10;

% Initialize sensor data
    Sensor_posit = [ 100 * 1000;      % sensor is 100 km in x
                    100 * 1000;      % sensor is 100 km in y
                    0 * 1000];      % sensor is 0 km in z

    sigma_r = 10;                    % 10 meters std dev in range
    sigma_b = 1*pi/180;              % 1 degree std dev in azimuth
    sigma_e = 1*pi/180;              % 1 degree std dev in elevation

    R = diag([sigma_r^2,              % covariance matrix for
              sigma_b^2,              % range and bearing measurements
              sigma_e^2]);

% Define the H matrix (MEASUREMENT MATRIX) for the accelerating
% model

    H = [1, 0, 0, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 1, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 0, 0, 1, 0, 0];

```

```

%*****
% ACCELERATING MODEL
%*****
% Define G matrix
    G_accel = -g * [0;
                     0;
                     0;
                     0;
                     0;
                     0;
                     (delta^2)/2;
                     delta;
                     0];

% Initialize Q, the covariance of the plant noise

    Q_sub_a = [(delta^5)/20, (delta^4)/8, (delta^3)/6;
                (delta^4)/8, (delta^3)/3, (delta^2)/2;
                (delta^3)/6, (delta^2)/2, delta ];

    Q_accel = q_sqr_a * [Q_sub_a, zeros(3), zeros(3);
                          zeros(3), Q_sub_a, zeros(3);
                          zeros(3), zeros(3), Q_sub_a ];

% Define F matrix (TRANSITION MATRIX) for discrete time
% accelerating model.

    f_sub_a = [1, delta, (delta^2)/2;
                0, 1, delta;
                0, 0, 1 ];

    F_accel = [f_sub_a, zeros(3), zeros(3);
                zeros(3), f_sub_a, zeros(3);
                zeros(3), zeros(3), f_sub_a ];

%*****
% BALLISTIC MODEL
%*****
% Define G matrix
    G_ball = -g * [0;
                    0;
                    0;
                    0;
                    0;
                    0;
                    (delta^2)/2;
                    delta;
                    0];

% Detemine Q for the Ballistic model

    Q_sub_b = [(delta^3)/3, (delta^2)/2, 0;
                (delta^2)/2, delta, 0;
                0, 0, 0];

```

```

    Q_ball = q_sqr_b * [ Q_sub_b, zeros(3), zeros(3);
                        zeros(3), Q_sub_b, zeros(3);
                        zeros(3), zeros(3), Q_sub_b];

% Define F matrix (TRANSITION MATRIX) for discrete time
% ballistic model.
    f_sub_b = [1, delta, 0;
               0, 1, 0;
               0, 0, 0];

    F_ball = [f_sub_b, zeros(3), zeros(3);
              zeros(3), f_sub_b, zeros(3);
              zeros(3), zeros(3), f_sub_b];

%***** End of Initialization outside loops *****

%*****
% Loop over the target motion/measurement simulation
%*****
    for kk = 1: nloops

tic
kk
        % define empty output matrices

        % measurement positions (cartesian) w/error
        zout_true_n = [];

        % distance error between measurement and true position
        error_true = [];

        % Kalman estimated trajectory
        K_track = [];
        K_accel = [];

        % error between Kalman track and actual track
        track_error = [];

%*****
% This block is used for the initialization process. Initialize
% from a SWAG.
%*****
        x_corr_accel = missilevec(2:10,1);
        P_corr_accel = eye(9) * 10^4;

        x_corr_ball = missilevec(2:10,1);
        P_corr_ball = eye(9) * 10^4;

        % Initial likelihoods for states.
        mu_init = [1;

```

```

0];

mu = mu_init;
mu_1 = mu(1);
mu_2 = mu(2);

%*****
% Loop through the simulation, generating target motion between
% sample times and taking measurements at each sample time,
% using 1 sensor
%*****
for ii = 2:nsamples

    % Process the measurement from Sensor

    % True missile position
    ztrue = [missilevec(2,ii);
             missilevec(5,ii);
             missilevec(8,ii)];

    %*****
    % convert current position to polar coordinates and add
    % sensor noise to the position, generating a noisy measurement
    % from the sensor.
    %*****

    % position relative to the sensor
    zrel = ztrue - Sensor_posit;

    r = sqrt(zrel(1)^2 + zrel(2)^2 + zrel(3)^2); % range
    from sensor
    b = atan2(zrel(2), zrel(1)); % bearing
    from sensor
    r_prime = sqrt(zrel(1)^2 + zrel(2)^2); % range in
    x/y plane
    e = atan2(zrel(3), r_prime); % elevation
    from sensor

    % add noise to the measurement
    r_n = r + sigma_r * randn;
    b_n = b + sigma_b * randn;
    e_n = e + sigma_e * randn;

    % measurement in polar + noise
    z_polar_n = [r_n;
                 b_n;
                 e_n];

    % measurement in cartesian coordinates + noise
    z_cart_rel_n = [r_prime*cos(b_n);
                   r_prime*sin(b_n);
                   r_n*sin(e_n) ];

    z_cart_true_n = z_cart_rel_n + Sensor_posit;

```

```

% compute measurement error in cartesian coordinates
zdiff = ztrue - z_cart_true_n;
disterror = sqrt(zdiff'*zdiff);

% Update the measurement array
% true cartesian measurement + error
zout_true_n = [zout_true_n, z_cart_true_n];

% measurement error (between true measurements)
error_true = [error_true, disterror];

%*****
% Prediction
%*****

% Probabilities of changing state, Markov chain
transition
    p1 = 1;
    p2 = 0.3;
    alt = 30e3;
    h = z_cart_true_n(3);

    prob_accel = -p2*( 1/(1+exp(-.0005*(h-alt))) - (1+p1) );
    prob_ball = 1 - prob_accel;

    rho = [prob_accel, prob_ball;
           0,          1 ];

% Accelerating Model
cbar = rho' * mu;

blowing up
    if cbar(1) < 10^(-8) % prevents cbar_1 from
        cbar_1 = 10^(-8);
    else
        cbar_1 = cbar(1);
    end;

    cbar_2 = cbar(2);

    rho_11 = rho(1,1);
    rho_21 = rho(2,1);
    rho_12 = rho(1,2);
    rho_22 = rho(2,2);

    x_hat_01 = x_corr_accel * ((rho_11*mu_1)/cbar_1) + ...
               x_corr_ball * ((rho_21*mu_2)/cbar_1);

    xtilde_11 = x_corr_accel - x_hat_01;
    xtilde_21 = x_corr_ball - x_hat_01;

    mu_11 = rho_11 * mu_1 / cbar_1;
    mu_21 = rho_21 * mu_2 / cbar_1;

```

```

P_hat_01 = mu_11 * (P_corr_accel + xtilde_11*xtilde_11') + ...
          mu_21 * (P_corr_ball + xtilde_21*xtilde_21');

% Kalman Filter Prediction Equations for Accelerating model
x_predict_accel = F_accel * x_hat_01 + G_accel;
P_predict_accel = F_accel * P_hat_01 * F_accel' + Q_accel;

% Ballistic Model
x_hat_02 = x_corr_accel * ((rho_12*mu_1)/cbar_2) + ...
          x_corr_ball * ((rho_22*mu_2)/cbar_2);

xtilde_12 = x_corr_accel - x_hat_02;
xtilde_22 = x_corr_ball - x_hat_02;

mu_12 = rho_12 * mu_1 / cbar_2;
mu_22 = rho_22 * mu_2 / cbar_2;

P_hat_02 = mu_12*(P_corr_accel + xtilde_12*xtilde_12') + ...
          mu_22*(P_corr_ball + xtilde_22*xtilde_22');

% Kalman Filter Prediction Equations for Ballistic model
x_predict_ball = F_ball * x_hat_02 + G_ball;
P_predict_ball = F_ball * P_hat_02 * F_ball' + Q_ball;

%*****
% Correction
%*****
% Accelerating Model
% Convert to relative position to compute polar coordinates
x_1 = x_predict_accel(1) - Sensor_posit(1);
x_4 = x_predict_accel(4) - Sensor_posit(2);
x_7 = x_predict_accel(7) - Sensor_posit(3);

% Convert prediction to polar coordinates
r_hat_a = sqrt(x_1^2 + x_4^2 + x_7^2);
b_hat_a = atan2(x_4, x_1);
e_hat_a = atan2(x_7, sqrt(x_1^2 + x_4^2));

% Determine expected measurement
z_polar_hat_a = [r_hat_a;
                 b_hat_a;
                 e_hat_a];

% Observed minus expected measurements
z_tilde_a = z_polar_n - z_polar_hat_a;

% The gradient of H evaluated at the most recent estimate
Hk_r2c1 = -x_4/(x_1^2 + x_4^2);
Hk_r2c4 = x_1/(x_1^2 + x_4^2);
Hk_r3c1 = (-x_1*x_7)/((sqrt(x_1^2 + x_4^2))*(x_1^2 + x_4^2 + x_7^2));
Hk_r3c4 = (-x_4*x_7)/((sqrt(x_1^2 + x_4^2))*(x_1^2 + x_4^2 + x_7^2));
Hk_r3c7 = (sqrt(x_1^2 + x_4^2))/(x_1^2 + x_4^2 + x_7^2);

```



```

% Determine H matrix
Hk_a = [x_1/r_hat_a, 0, 0, x_4/r_hat_a, 0, 0, x_7/r_hat_a, 0, 0;
        Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
        Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

% Compute Kalman Gain
K_accel = P_predict_accel*Hk_a' * inv(Hk_a * P_predict_accel * Hk_a'+R);

% Kalman Filter Correction equations for Accelerating Model
x_corr_accel = x_predict_accel + K_accel * z_tilde_a;
P_corr_accel = (eye(9) - K_accel*Hk_a)* P_predict_accel;

% Ballistic Model
% Convert to relative position to compute polar coordinates
x_1 = x_predict_ball(1) - Sensor_posit(1);
x_3 = x_predict_ball(4) - Sensor_posit(2);
x_5 = x_predict_ball(7) - Sensor_posit(3);

% Convert prediction to polar coordinates
r_hat_b = sqrt(x_1^2 + x_3^2 + x_5^2);
b_hat_b = atan2(x_3, x_1);
e_hat_b = atan2(x_5, sqrt(x_1^2 + x_3^2));

% Determine expected measurement
z_polar_hat_b = [r_hat_b;
                 b_hat_b;
                 e_hat_b];

% Observed minus expected measurements
z_tilde_b = z_polar_n - z_polar_hat_b;

% The gradient of H evaluated at the most recent estimate
Hk_r2c1 = -x_3/(x_1^2 + x_3^2);
Hk_r2c4 = x_1/(x_1^2 + x_3^2);
Hk_r3c1 = (-x_1*x_5)/( (sqrt(x_1^2 + x_3^2))*(x_1^2 + x_3^2 + x_5^2) );
Hk_r3c4 = (-x_3*x_5)/( (sqrt(x_1^2 + x_3^2))*(x_1^2 + x_3^2 + x_5^2) );
Hk_r3c7 = (sqrt(x_1^2 + x_3^2))/(x_1^2 + x_3^2 + x_5^2);

% Determine H matrix
Hk_b = [x_1/r_hat_b, 0, 0, x_3/r_hat_b, 0, 0, x_5/r_hat_b, 0, 0;
        Hk_r2c1, 0, 0, Hk_r2c4, 0, 0, 0, 0, 0;
        Hk_r3c1, 0, 0, Hk_r3c4, 0, 0, Hk_r3c7, 0, 0];

% Compute Kalman Gain
K_ball = P_predict_ball * Hk_b'*inv(Hk_b*P_predict_ball * Hk_b' + R);

% Kalman Filter Correction equations for the Ballistic Model
x_corr_ball = x_predict_ball + K_ball * z_tilde_b;
P_corr_ball = (eye(9) - K_ball*Hk_b)* P_predict_ball;

%*****
% Update mode probabilities
%*****
m = 3;

```

```

        S_1 = Hk_a * P_predict_accel * Hk_a' + R;
lambda_1 = (exp(-(z_tilde_a)'*inv(S_1)*z_tilde_a/2))/(sqrt((2*pi)^m *
det(S_1)));

        S_2 = Hk_b * P_predict_ball * Hk_b' + R;
lambda_2 = (exp(-(z_tilde_b)'*inv(S_2)*z_tilde_b/2))/(sqrt((2*pi)^m *
det(S_2)));

        c = lambda_1 * cbar_1 + lambda_2 * cbar_2;

        mu_1 = lambda_1 * cbar_1/c;
        mu_2 = lambda_2 * cbar_2/c;

        mu = [mu_1;
              mu_2];

%*****
% Produce Combined Estimates
%*****
        x_corr = mu_1 * x_corr_accel + mu_2 * x_corr_ball;
        P_corr = mu_1*(P_corr_accel+(x_corr_accel-
x_corr)*(x_corr_accel-x_corr)')+...
              mu_2*(P_corr_ball +(x_corr_ball-
x_corr)*(x_corr_ball- x_corr)');

%*****

and        % Kalman track positions and difference between Kalman

        % actual track position and actual target position
        zout_K_track = H*x_corr;

        track_diff = ztrue - zout_K_track;
        track_error = [track_error, sqrt(track_diff'*track_diff)];

        % Update KF track trajectory array
        K_track = [K_track, zout_K_track];

end; % for ii = 2:20:nsamples

%*****

        if kk == 1,                % create first output

                zoutmean_true = zout_true_n;
                mean_K_track = K_track;
                merror_track = track_error;
                merror = error_true;

        else                        % create output after 1st run

```

```

        zoutmean_true = zoutmean_true + zout_true_n;
        mean_K_track = mean_K_track + K_track;
        merror_track = merror_track + track_error;
        merror = merror + error_true;

    end; % if kk ==1, else

toc

end; % for kk = 1:nloops

%*****
% Compute Means
%*****
zoutmean_true = zoutmean_true/nloops;
mean_K_track = mean_K_track/nloops;
merror = merror/nloops; % mean error between
                        % measurement and true position

merror_track = merror_track/nloops; % mean error between
                                % EKF estimated position
                                % and true position

%*****
% Plot results
%*****
figure(1)
    measurement = zoutmean_true/1000; % convert to km
    Kalman_track = mean_K_track/1000; % convert to km
    missile_track = missilevec(:,1:nsamples)/1000; % convert to km

    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');%,...
        %Sensor_posit(1)/1000, Sensor_posit(2)/1000,
Sensor_posit(3)/1000, 'rx');
    %axis('equal');
    %axis([0,40,0,40,0,40]);
    axis([0,35,0,35,0,35])
    title(['TBM Profile ', num2str(prof_num)]);
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
% print -deps imm3a

figure(2)
    plot3(missile_track(2,:), missile_track(5,:), missile_track(8,:), 'g-');%,...
        measurement(1,:), measurement(2,:), measurement(3,:), 'r-');%,...

%Sensor_posit(1)/1000,Sensor_posit(2)/1000,Sensor_posit(3)/1000, 'rx');
%axis('equal')
%axis([0,40,0,40,0,40]);
axis([0,35,0,35,0,35])
title(['TBM Profile ', num2str(prof_num), ' w/ Measurement Noise']);

```

```

    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
% print -deps imm3b

figure(3)
    plot3(missile_track(2,1:nsamples), missile_track(5,1:nsamples),
missile_track(8,1:nsamples),'g-',...
        Kalman_track(1,:), Kalman_track(2,:), Kalman_track(3,:),'r-
');%,...

%Sensor_posit(1)/1000,Sensor_posit(2)/1000,Sensor_posit(3)/1000,'rx');
%axis('equal')
%axis([0,40,0,40,0,40]);
axis([0,35,0,35,0,35])
    xlabel('X (km)'), ylabel('Y (km)'), zlabel('Z (km)'),grid;
    title(['TBM Profile ', num2str(prof_num),' w/ IMM Trajectory']);
% print -deps imm3c

figure(4)
    time = missilevec(1,:);
    plot(time(1:nsamples-1), merror,'g-', time(1:nsamples-1),
merror_track,'r-');
    xlabel('Time (seconds)'),ylabel('Mean Error (meters)'),grid;
    title('Mean Distance Error in Measurements vs Time');
%print -deps imm3d

%save mm5003
%save imm1003

```

APPENDIX F. MATLAB[®] INFORMATION

MATLAB[®] and SIMULINK[™] is a product of MathWorks, Inc., 24 Prime Way, Natick, Mass. 01760. MATLAB[®] version 4.2b and SIMULINK[™] version 1.3a were used throughout this study.

LIST OF REFERENCES

1. Senator Jesse Helms, speech on the Strategic Anti-Missile Revitalization Act of 1996, U.S. Senate, 104th Cong., *Congressional Record* (6 February 1996), S 917.
2. Mosher, D., "The Grand Plans," *IEEE Spectrum*, Vol.34, No. 9, September 1997.
3. Greenburg, J., "Theatre Ballistic Missile Defense: New United States Strategic Requirements and the ABM Treaty," Master's Thesis, Naval Postgraduate School, California, 1995.
4. Isaacson, J., and Vaughan, D., *Estimation and Prediction of Ballistic Missile Trajectories*, RAND, Santa Monica, California, 1996.
5. Stevens, B., and Lewis, F., *Aircraft Control and Simulation*, John Wiley and Sons, Inc., New York, 1992.
6. Zarchan, P., *Tactical and Strategic Missile Guidance, Second Edition*, Artech House, Inc., Norwood, Massachusetts, 1986.
7. Blackman, S., *Multiple-Target Tracking with Radar Application*, American Institute of Aeronautics and Astronautics, Inc., Washington, D.C., 1994.
8. Brown, R., and Hwang, P., *Introduction to Random Signals and Applied Kalman Filtering, Third Edition*, John Wiley and Sons, New York, 1997.
9. Bar-Shalom, Y., and Li, X., *Estimation and Tracking: Principles, Techniques, and Software*, Artech House, Inc., Norwood, Massachusetts, 1993.
10. Bar-Shalom, Y., and Li, X., *Multitarget-Multisensor Tracking: Principles and Techniques*, Artech House, Inc., Norwood, Massachusetts, 1995.
11. Hutchins, R.G., EC3310 Class Notes, Naval Postgraduate School, 1997.
12. Jerardi, T., TBM Profile Data, Johns Hopkins University Applied Physics Laboratory, 1998.
13. Beaulieu, M., "Launch Detection Satellite System Engineering Error Analysis," Master's Thesis, Naval Postgraduate School, California, 1996.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center..... 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2. Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4. Professor Robert G. Hutchins, Code EC/Hu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
5. Professor Harold A Titus, Code EC/Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
6. Fred Glaeser Department of the Navy, CNO N632, Room 5P773 Pentagon Washington, DC 20350-2000	1
7. Dr. Verle N. Schrodtt Associate Dean for Administration College of Engineering University of Alabama Box 870200 Tuscaloosa, AL 35487-0200	1
8. LT Antonio P. San Jose 7901 Allentown Rd. Fort Washington, MD 20744	2